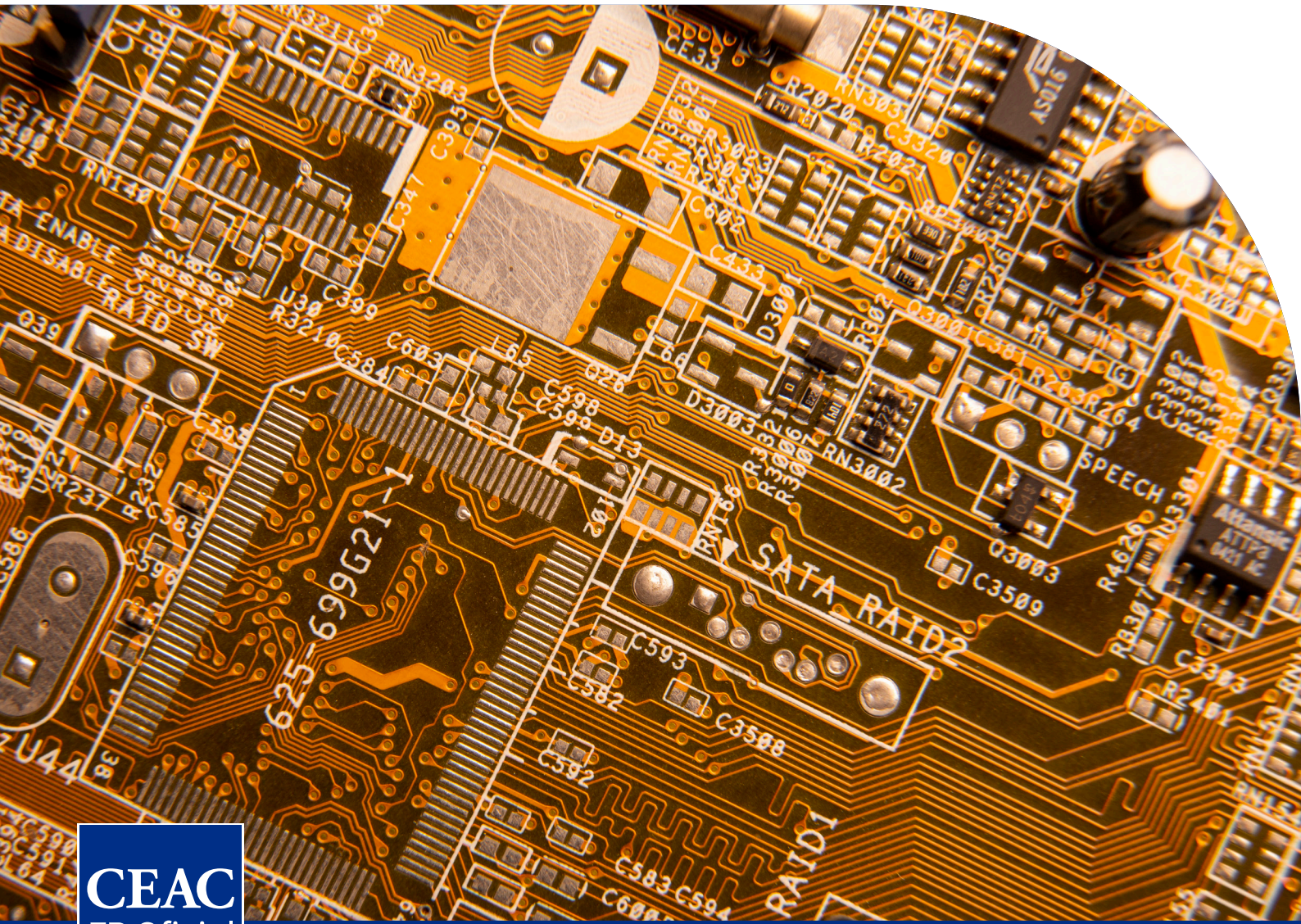
Desarrollo de aplicaciones multiplataforma

MÓDULO:

Base de datos

UNIDAD:

Base de datos relacionales



CEAC
FP Oficial

MÓDULO:

Base de datos

UNIDAD:

Base de datos relacionales

CEAC FP Oficial

ÍNDICE

1. Introducción

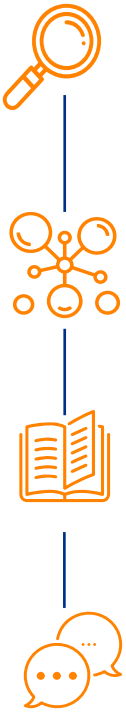
A lo largo de esta unidad estudiaremos los conceptos básicos de una base de datos relacional. Aprenderemos en qué consiste el modelo relacional y conoceremos su terminología más frecuente.

El **modelo relacional** de una base de datos fue introducido en 1970 por el informático inglés Edgar Frank Codd, en los laboratorios IBM, en San José (California), y no tardó en consolidarse como un nuevo paradigma en los modelos de bases de datos. Se trata de una estructura de datos simples y uniformes (la relación), es decir, de una nueva forma de estructurar y procesar una base de datos, en la que el lugar y el modo en como se almacenen los datos no tienen relevancia.

La ventaja del modelo relacional es que los datos se almacenan, al menos conceptualmente, de una manera que a los usuarios les resulta más fácil de entender y utilizar. Los datos se almacenan como tablas, lo que permite establecer interconexiones (relaciones) entre los datos de las distintas tablas de una forma muy versátil y sencilla y, a través de dichas conexiones, relacionar los datos de ambas tablas, motivo por el cual se le conoce con el nombre de modelo relacional. Durante su diseño, una base de datos relacional pasa por un proceso llamado **normalización de una base de datos.**

2. Objetivos pedagógicos

Los objetivos de aprendizaje que se pretenden alcanzar con esta unidad son:



- **Comprender los conceptos básicos del modelo relacional.**
- **Familiarizarse con la terminología del modelo relacional.**
- **Reconocer la importancia histórica del modelo relacional.**
- **Aplicar el concepto de almacenamiento y procesamiento de datos en tablas.**

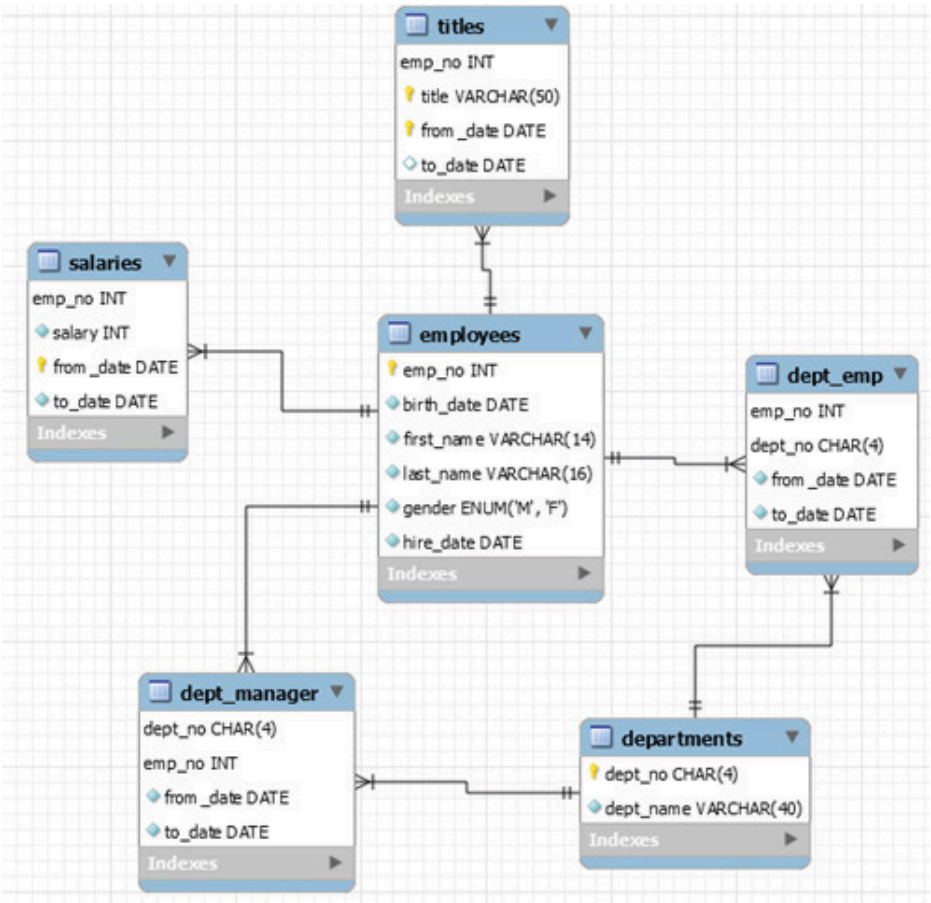
3. Base de datos relacionales

3.1 Modelo de datos

El **modelo de datos** es un lenguaje orientado a describir una base de datos. Un modelo de datos permite describir:

- **Estructuras de datos.** Son los tipos de datos contenidos en la base y la forma en que éstos se relacionan.
- **Restricciones de integridad.** Son las condiciones que los datos deben cumplir para reflejar correctamente la realidad deseada.
- **Operaciones de manipulación.** Son aquellas operaciones de agregado, borrado, modificación y recuperación de los datos de la base.

En la siguiente figura puede verse claramente la estructura que sigue la base de datos que representa, es decir, las relaciones que existen entre las diferentes tablas, mediante sus correspondientes fl echas, así como las llaves que indican qué atributos son claves primarias de la tabla. Si nos fijamos, veremos que al lado de cada atributo está el tipo de dato que almacena (enteros, caracteres, boleanos, fechas, etc.), lo que nos ayudará a la hora de saber qué datos debemos introducir en la base de datos.



Ejemplo de modelo de datos.

La clasificación de los modelos de datos se realiza según el nivel de abstracción que presentan:

- **Modelo conceptual:** También conocido como modelo de datos de alto nivel, representa los conceptos y las relaciones entre ellos en un nivel abstracto. Este modelo se centra en la estructura lógica de la información y no en los detalles de implementación. El modelo conceptual proporciona una visión global de la base de datos y se utiliza para diseñar y comunicar la estructura de datos a los stakeholders y usuarios. Ejemplos de modelos conceptuales son el Modelo Entidad-Relación (ER) y el Modelo de Objetos.

Conceptual = estructura

Lógico = define la forma en la que se almacenan y acceden los datos en la base

Físico = estructura de archivos, índices, particiones y técnicas de optimización

- **Modelo lógico:** También llamado modelo de datos intermedio, representa la estructura de la base de datos en un nivel más detallado que el modelo conceptual. Este modelo se enfoca en la organización y representación lógica de los datos en términos de tablas, relaciones y restricciones. El modelo lógico define la forma en que se almacenan y acceden los datos en la base de datos. Ejemplos de modelos lógicos son el Modelo Relacional y el Modelo de Red.

- **Modelo físico:** Es el modelo de datos de nivel más bajo y se centra en los detalles de implementación física de la base de datos en el sistema de almacenamiento. Define cómo se almacenan los datos en el disco y cómo se accede a ellos de manera eficiente. El modelo físico se preocupa por aspectos como la estructura de archivos, los índices, las particiones y las técnicas de optimización. Ejemplos de modelos físicos son el Modelo de Almacenamiento Columnar y el Modelo de Almacenamiento en Árbol B.

3.2 Terminología del modelo relacional

El **modelo relacional** representa la base de datos como una colección de relaciones. La terminología asociada con el modelo relacional es la siguiente:

- **Tablas**: En el modelo relacional, una tabla es una estructura fundamental que representa una entidad o concepto del mundo real. Las tablas están compuestas por filas y columnas, y se utilizan para almacenar y organizar los datos. Cada tabla tiene un nombre único que la identifica y está formada por un conjunto de atributos.

- **Atributos**: Los atributos son las características o propiedades que describen una entidad u objeto en un modelo de datos. Definen qué información se almacenará y cómo se representará en la base de datos. Los atributos permiten organizar y filtrar los datos, y son fundamentales para la manipulación y consulta de la información en una base de datos.

- **Filas y columnas:** Una fila, también conocida como tupla, es una instancia o registro en una tabla. Cada fila contiene datos específicos relacionados con los atributos definidos en la tabla. Cada atributo está representado por una columna en la tabla, y cada columna tiene un nombre único y un tipo de dato asociado.

RECUERDA: En la terminología del modelo relacional, una tabla se denomina relación, una tupla es una fila y un atributo es una columna. A las entidades también se las conoce como una relación.

Los términos relación y tupla han caído en desuso. Así pues, utilizaremos para ellos la terminología más común y denominaremos a la relación **tabla** (table) y **fila** (row) a la **tupla**. En cambio, el término atributo, denominado también **columna** (column), se utiliza indistintamente al hablar del modelo relacional.

Modelo relacional	Sistemas gestores de bases de datos	Sistemas gestores de bases de datos
Relación Tupla Atributo	Tabla Fila Columna	Fichero Registro Campo

Tabla de equivalencias en la terminología.

3.2.1 Relaciones y cardinalidad

En el modelo relacional, las relaciones establecen vínculos entre diferentes tablas utilizando claves primarias y claves ajenas. La cardinalidad se refiere a la cantidad de instancias o registros relacionados entre dos tablas. Se utilizan varios tipos de relaciones, como:

- **Relación uno a uno (1:1):** Una instancia de una tabla se relaciona con exactamente una instancia de otra tabla.
- **Relación uno a muchos (1:N)**: Una instancia de una tabla se relaciona con múltiples instancias de otra tabla.
- **Relación muchos a muchos (N:M):** Múltiples instancias de una tabla se relacionan con múltiples instancias de otra tabla.

PARA SABER MÁS: El número de filas de una tabla se denomina cardinalidad de la tabla; el número de columnas es el grado de la tabla; y el conjunto de valores válidos que puede tener cada atributo se llama dominio.

3.3 Características de una relación

Uno de los logros del modelo relacional radica en la simplicidad del diseño básico de la tabla. Veamos cuáles son sus características:

- **A cada tabla se le da un nombre.** No pueden existir dos tablas con el mismo nombre. Cada tabla es, a su vez, un conjunto de filas y columnas.
- **La información en la base de datos son datos explícitos**. No existen apuntadores o ligas entre tablas.

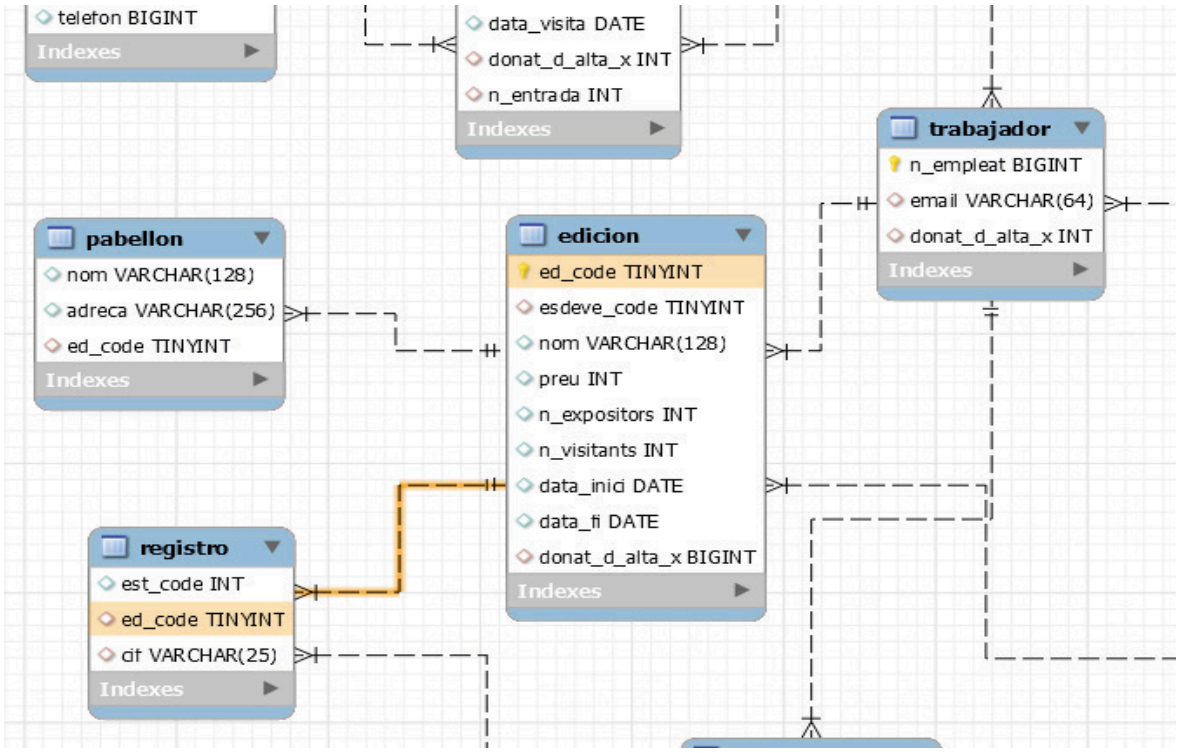
- **Cada fila de la tabla debe ser única**, es decir, no pueden existir dos filas con datos idénticos para todos los atributos. La razón es que cada fila representa un hecho diferenciado, y la presencia de dos filas idénticas sería redundante.
- El **orden de las filas** de una tabla es irrelevante.
- Todas las **entradas** de cualquier columna son de un solo tipo.
- A cada columna **se le asigna un nombre** que refleje la característica correspondiente del mundo real.
- El **orden de las columnas** no es importante para la tabla.
- Cada **entrada de una tabla** debe tener un sólo valor (son atómicos); no se admiten valores múltiples. En definitiva, la intersección determinada de fila y columna tiene un solo valor, nunca un conjunto de valores.

2.3.1 Reglas de integridad de las relaciones

Las reglas de integridad aseguran la consistencia y validez de los datos almacenados en las relaciones. Estas reglas se aplican para mantener la integridad referencial, evitar redundancias y garantizar la consistencia de los datos. Algunas reglas comunes incluyen:

- **Clave primaria**: Cada relación debe tener una clave primaria única que identifique de manera única cada fila en la relación.
- **Clave foránea:** Permite establecer relaciones entre diferentes relaciones al referenciar la clave primaria de una relación en otra relación.
- **Restricciones de dominio**: Establecen los límites y formatos válidos para los valores de los atributos.
- **Restricciones de integridad referencial:** Garantizan que las relaciones entre las tablas sean válidas y consistentes.

Dentro del modelo relacional existe una relación entre las diferentes tablas que ayudan a evitar los datos redundantes. Esta relación vincula la clave primaria de una tabla, que proporciona a un identificador único para cada fila, con una clave ajena o foránea de la otra tabla.



Ejemplo de una tabla y sus vinculaciones.

A continuación, podremos ver la relación que existe entre la Tabla director y la Tabla empleados.

Cada empleado tiene un director asociado mediante la columna "Director", que es clave foránea de la clave primaria DNI_director en la otra tabla. De esta manera, se crea una dependencia de la tabla foránea, y así, si eliminamos al director, Sergio Martínez, el registro del empleado Fernando García será incorrecto, puesto que estará referenciado a un elemento inexistente lo que viola la integridad referencial.

3.4 Tipos de datos

La teoría de bases de datos relacionales no establece ninguna norma concreta con respecto a los **tipos de datos** que éstas deben contener. Por lo tanto, cualquier conjunto de valores disponibles en una determinada plataforma de hardware y sistema operativo, o incluso los nuevos tipos de datos defi nidos por el usuario, podrían ser tipos de datos válidos.

Los **sistemas gestores de bases de datos** incorporan sus propias especificaciones en cuanto a tipos de datos. Para cada columna de cada una de las tablas, es necesario determinar el tipo de datos que debe contener, de esa forma, se ajusta el diseño de la base de datos y se consigue un almacenamiento óptimo con el mínimo espacio.

Los tipos de datos numéricos se utilizan para almacenar valores numéricos, como enteros y números decimales. Algunos ejemplos de tipos de datos numéricos son:

- **Entero (INTEGER):** Almacena números enteros sin decimales, como 1, 2, -3, etc.
- **Decimal (DECIMAL):** Almacena números decimales de precisión fija, como 3.14, 10.50, -0.75, etc.
- **Real (REAL):** Almacena números en coma flotante de precisión simple.
- **Doble precisión (DOUBLE):** Almacena números en coma flotante de doble precisión.
- **Texto**:Los tipos de datos de texto se utilizan para almacenar cadenas de caracteres y representar información de texto (Figura 2.6). Algunos ejemplos de tipos de datos de texto son:
  - **Cadena de caracteres (VARCHAR):** Almacena cadenas de longitud variable.
  - **Texto (TEXT)**: Almacena textos más extensos, con longitud variable.
  - **Carácter (CHAR)**: Almacena caracteres individuales con longitud fija.
  - **Otros tipos de datos comunes:**

  Además de los tipos de datos numéricos, de texto y de fecha y hora, existen otros tipos de datos comunes que se utilizan en las bases de datos, como:

  - **Booleano (BOOLEAN):** Almacena valores verdaderos o falsos.
  - **Binario (BINARY):** Almacena datos binarios, como imágenes o archivos.
  - **Enumeración (ENUM):** Almacena un conjunto predefinido de valores posibles.
  - **Objeto grande (LOB)**: Almacena objetos grandes, como archivos multimedia.

Tipo de dato	Características
CHAR	Cadena de caracteres de longitud fija de caracteres. El tamaño máximo es de 2000 bytes o caracteres.
NCHAR	Cadena de caracteres Unicode de longitud fija similar al char. El tamaño máximo está determinado por la definición del juego de carácteres.
VARCHAR2	Cadena de caracteres de longitud variable. El tamaño máximo es de 4000 bytes o caracteres, y la mínima es de 1 byte o un carácter.
NVARCHAR2	Cadena de caracteres Unicode de longitud variable similar al varchar2. El tamaño máximo está determinado por la definición del juego de carácteres.
NUMBER	Datos numéricos con enteros y decimales con o sin signo.
FLOAT	Un subtipo del tipo de datos NUMBER.
DATE	Este tipo de datos contiene los campos de fecha y hora. Año, mes, día, hora, minuto y segundo.
TIMESTAMP	Este tipo de datos contiene los campos datetime. Año, mes, día, hora, minuto y segundo. Contiene las fracciones de segundo, pero no tiene una zona horaria.
TIMESTAMP ... WITH TIME ZONE	Este tipo de datos contiene los campos datetime. Año, mes, día, hora, minuto, segundo, TIMEZONE_HOUR y TIMEZONE_MINUTE. Cuenta con las fracciones de segundo y una zona horaria explícita.
ROWID	Cadena que representa la dirección única de una fila en la tabla.
RAW	Datos binarios. El tamaño máximo es de 2000 bytes.
LONG RAW	Datos binarios de longitud variable. Se utiliza para almacenar gráficos, sonidos, etc. El tamaño máximo es de 2 gigabytes.
CLOB	Un objeto que contiene caracteres. Son compatibles tanto de ancho fijo y conjuntos de ancho variable de caracteres, con el carácter de base de datos establecida. El tamaño máximo es (4 gigabytes - 1) * (tamaño del bloque de la base de datos).
NCLOB	Un objeto que contiene caracteres Unicode. El tamaño máximo está determinado por la definición del juego de carácteres.
BLOB	Un objeto de tipo binario. El tamaño máximo es (4 gigabytes - 1) * (tamaño del bloque de la base de datos).

Tipos de datos más usuales.

3.5 Juegos de caracteres

Se llama **juego de caracteres** (charset) al conjunto de caracteres reconocidos por un hardware o un software. Cada código es una representación numérica de un determinado carácter. Por ejemplo, el conjunto de caracteres ASCII (American Standard Code for Information Interchange) está basado en el alfabeto latino, tal como se utiliza en inglés.

ASCII es un código de siete bits, lo que significa que utiliza cadenas de bits con siete dígitos binarios (que van de 0 a 127 en base decimal) para representar información de caracteres.

RECUERDA: Al empezar a trabajar con una base de datos determinada, es conveniente conocer las especificaciones del tipo de datos que ésta puede contener.

Existen extensiones que utilizan 8 bits para proporcionar caracteres adicionales, que son utilizados en idiomas distintos del inglés. En juegos de caracteres de 8 bits, podemos representar hasta 256 caracteres diferentes, como, por ejemplo, UTF-8 o ANSI. Actualmente, se utilizan juegos de caracteres de 16 bits, que permiten codificar hasta 65.535 caracteres; es el caso de Unicode. Casi todos los Bases de datos actuales utilizan el código ASCII o una extensión compatible.

En definitiva, un juego de caracteres es un sistema de codificación para todos los caracteres y signos que utilizamos en la representación y comunicación de la información. Su importancia se debe a que determina qué caracteres pueden presentarse en nuestro sistema y pueden alterar la presentación de los datos; adicionalmente, también pueden alterar el criterio de ordenación alfabética.

Caracteres de control ASCII

DEC	HEX	Símbolo ASCII	
00	00h	NULL	(carácter nulo)
01	01h	SOH	(inicio encabezado)
02	02h	STX	(inicio texto)
03	03h	ETX	(fin de texto)
04	04h	EOT	(fin transmisión)
05	05h	ENQ	(enquiry)
06	06h	ACK	(acknowledgement)
07	07h	BEL	(timbre)
08	08h	BS	(retroceso)
09	09h	HT	(tab horizontal)
10	0Ah	LF	(salto de linea)
11	0Bh	VT	(tab vertical)
12	0Ch	FF	(form feed)
13	0Dh	CR	(retorno de carro)
14	0Eh	SO	(shift Out)
15	0Fh	SI	(shift In)
16	10h	DLE	(data link escape)
17	11h	DC1	(device control 1)
18	12h	DC2	(device control 2)
19	13h	DC3	(device control 3)
20	14h	DC4	(device control 4)
21	15h	NAK	(negative acknowle.)
22	16h	SYN	(synchronous idle)
23	17h	ETB	(end of trans. block)
24	18h	CAN	(cancel)
25	19h	EM	(end of medium)
26	1Ah	SUB	(substitute)
27	1Bh	ESC	(escape)
28	1Ch	FS	(file separator)
29	1Dh	GS	(group separator)
30	1Eh	RS	(record separator)
31	1Fh	US	(unit separator)
127	20h	DEL	(delete)

Caracteres ASCII imprimibles

DEC	HEX	Símbolo	DEC	HEX	Símbolo	DEC	HEX	Símbolo
32	20h	espacio	64	40h	@	96	60h	`
33	21h	!	65	41h	A	97	61h	a
34	22h	"	66	42h	B	98	62h	b
35	23h	#	67	43h	C	99	63h	c
36	24h	$	68	44h	D	100	64h	d
37	25h	%	69	45h	E	101	65h	e
38	26h	&	70	46h	F	102	66h	f
39	27h	'	71	47h	G	103	67h	g
40	28h	(	72	48h	H	104	68h	h
41	29h	)	73	49h	I	105	69h	i
42	2Ah	*	74	4Ah	J	106	6Ah	j
43	2Bh	+	75	4Bh	K	107	6Bh	k
44	2Ch	,	76	4Ch	L	108	6Ch	l
45	2Dh	-	77	4Dh	M	109	6Dh	m
46	2Eh	.	78	4Eh	N	110	6Eh	n
47	2Fh	/	79	4Fh	O	111	6Fh	o
48	30h	0	80	50h	P	112	70h	p
49	31h	1	81	51h	Q	113	71h	q
50	32h	2	82	52h	R	114	72h	r
51	33h	3	83	53h	S	115	73h	s
52	34h	4	84	54h	T	116	74h	t
53	35h	5	85	55h	U	117	75h	u
54	36h	6	86	56h	V	118	76h	v
55	37h	7	87	57h	W	119	77h	w
56	38h	8	88	58h	X	120	78h	x
57	39h	9	89	59h	Y	121	79h	y
58	3Ah	:	90	5Ah	Z	122	7Ah	z
59	3Bh	;	91	5Bh	[	123	7Bh	{
60	3Ch	<	92	5Ch	\	124	7Ch	|
61	3Dh	=	93	5Dh	]	125	7Dh	}
62	3Eh	>	94	5Eh	^	126	7Eh	~
63	3Fh	?	95	5Fh	_			

ASCII extendido

DEC	HEX	Símbolo	DEC	HEX	Símbolo	DEC	HEX	Símbolo	DEC	HEX	Símbolo
128	80h	Ç	160	A0h	á	192	C0h	└	224	E0h	Ó
129	81h	ü	161	A1h	í	193	C1h	┴	225	E1h	ß
130	82h	é	162	A2h	ó	194	C2h	┬	226	E2h	Ô
131	83h	â	163	A3h	ú	195	C3h	├	227	E3h	Ò
132	84h	ä	164	A4h	ñ	196	C4h	─	228	E4h	õ
133	85h	à	165	A5h	Ñ	197	C5h	┼	229	E5h	Õ
134	86h	å	166	A6h	ª	198	C6h	ã	230	E6h	µ
135	87h	ç	167	A7h	º	199	C7h	Ã	231	E7h	þ
136	88h	ê	168	A8h	¿	200	C8h	╚	232	E8h	Þ
137	89h	ë	169	A9h	®	201	C9h	╔	233	E9h	Ú
138	8Ah	è	170	AAh	¬	202	CAh	╩	234	EAh	Û
139	8Bh	ï	171	ABh	½	203	CBh	╦	235	EBh	Ù
140	8Ch	î	172	ACh	¼	204	CCh	╠	236	ECh	ý
141	8Dh	ì	173	ADh	¡	205	CDh	═	237	EDh	Ý
142	8Eh	Ä	174	AEh	«	206	CEh	╬	238	EEh	¯
143	8Fh	Å	175	AFh	»	207	CFh	¤	239	EFh	´
144	90h	É	176	B0h	░	208	D0h	ð	240	F0h	≡
145	91h	æ	177	B1h	▒	209	D1h	Ð	241	F1h	±
146	92h	Æ	178	B2h	▓	210	D2h	Ê	242	F2h	‗
147	93h	ô	179	B3h	│	211	D3h	Ë	243	F3h	¾
148	94h	ò	180	B4h	┤	212	D4h	È	244	F4h	¶
149	95h	ò	181	B5h	Á	213	D5h	ı	245	F5h	§
150	96h	û	182	B6h	Â	214	D6h	Í	246	F6h	÷
151	97h	ù	183	B7h	À	215	D7h	Î	247	F7h	¸
152	98h	ÿ	184	B8h	©	216	D8h	Ï	248	F8h	°
153	99h	Ö	185	B9h	╣	217	D9h	┘	249	F9h	¨
154	9Ah	Ü	186	BAh	║	218	DAh	┌	250	FAh	·
155	9Bh	ø	187	BBh	╗	219	DBh	█	251	FBh	¹
156	9Ch	£	188	BCh	╝	220	DCh	▄	252	FCh	³
157	9Dh	Ø	189	BDh	¢	221	DDh	¦	253	FDh	²
158	9Eh	×	190	BEh	¥	222	DEh	Ì	254	FEh	■
159	9Fh	ƒ	191	BFh	┐	223	DFh	▀	255	FFh	

RECUERDA: Sólo se podrán almacenar correctamente en la base de datos los caracteres admitidos por el juego de caracteres definidos en cada base de datos.

3.6 Claves primarias

Una **clave candidata** es un atributo o conjunto de atributos que también cumple con las propiedades de unicidad y no nulidad, al igual que una clave primaria. Sin embargo, a diferencia de la clave primaria, una clave candidata no necesariamente ha sido seleccionada como la clave principal de una tabla.

La **clave primaria** es un atributo o conjunto de atributos que identifica de forma única cada registro en una tabla. Su propósito principal es garantizar la integridad de los datos y proporcionar una forma de identificar y acceder a registros específicos de manera eficiente. La clave primaria es una columna o combinación de columnas que no puede contener valores duplicados ni valores nulos.

Cada tabla puede tener **múltiples claves candidatas**, y es tarea del diseñador de la base de datos seleccionar una de ellas como clave primaria. La selección de una clave primaria se basa en consideraciones como la simplicidad, la estabilidad y la naturaleza inherente de los datos. Una vez que se ha elegido una clave primaria, las claves candidatas restantes se denominan claves alternativas y no se utilizan como identificadores principales de los registros.

DNI	Apellido	Nombre	Ciudad
00000000 A	Sánchez	Pablo	Barcelona
00000001 B	Martínez	Sergio	Madrid
00000002 C	Pérez	Álex	Barcelona

Listado de personas con diferentes atributos que las identifican.

La tabla representa un listado de personas y en ella podremos definir diferentes claves candidatas.

En ella definiremos tres claves candidatas:

- DNI
- Nombre
- Apellido

Cualquiera de estas tres claves candidatas podría identificar a una persona. Ahora bien, podemos suponer que pueden existir dos personas con el mismo nombre y apellido; en cambio, es imposible que existan dos personas con el mismo DNI. Por ello, en este caso lo más conveniente sería **definir el DNI como la clave primaria** de la tabla.

La **clave primaria** es aquella clave candidata que el usuario elegirá, por consideraciones ajenas al modelo relacional, para identificar las filas de la tabla; lo que la convierte en un identificador que será siempre único para cada fila. El modelo relacional no incluye el concepto de elegir una clave como clave primaria cuando hay varias claves candidatas. En ocasiones, el usuario decide crear una clave primaria numérica, de manera aleatoria, cuyo único requisito es el de guardar un valor numérico único para cada fila de forma totalmente independiente a los datos de negocio y que, generalmente, no tiene significado por sí misma. Este tipo de claves primarias también se conocen como **claves subrogadas** (surrogate key).

Diremos que una clave primaria es **simple** cuando el conjunto de atributos no vacío está compuesto por una sola columna. Sin embargo, una clave primariab se llamará clave **compuesta** cuando el conjunto de atributos no vacío este formado por más de una columna, creando siempre una sola clave única. Es decir, si se opta por utilizar una **clave compuesta**, ésta deberá ser única con respecto a las demás filas de la tabla.

Base de datos relacional

Tabla 1			
Columna 1	Columna 2	Columna 3	Columna 4
(Clave primaria)		(Clave alternativa)	

Tabla 2			
Columna 1	Columna 2	Columna 3	Columna 4
(Clave primaria compuesta)			

Tabla 3			
Columna 1	Columna 2	Columna 3	Columna 4
	(Clave alternativa)		(Clave subrogada)

Distintos tipos de claves primarias para una tabla.

Imaginemos ahora que, por cuestiones de seguridad, no deseamos conservar el DNI de las personas en nuestra base de datos; en ese caso, deberíamos buscar una alternativa. Por separado, tanto el nombre como el apellido (nuestras claves candidatas) pueden ser susceptibles de que existan repeticiones; en cambio, hay muchas menos posibilidades de que existan dos personas con el mismo nombre y apellido. Por tanto, defi niremos una clave primaria compuesta que agrupará las columnas **Nombre y Apellido** en una única clave.

Apellido	Nombre	Ciudad
Sánchez	Pablo	Pablo
Sánchez	Sergio	Sergio
Pérez	Pablo	Pablo

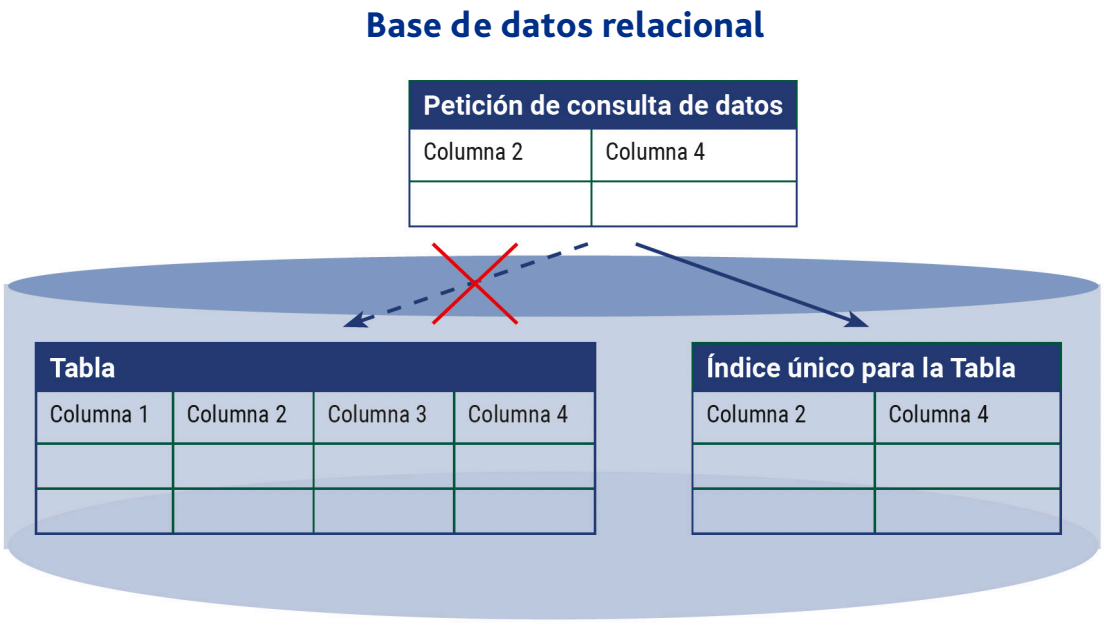
Tabla de personas con sus ciudades de origen.

RECUERDA:Sólo puede haber una clave primaria por tabla, pero ningún atributo de dicha clave puede contener valores vacíos (Null)

3.7 Índices

El **índice de una base de datos** mejora la velocidad de las consultas, lo que permite un rápido acceso a las filas de una tabla en una base de datos. Al aumentar drásticamente la velocidad de acceso, el índice de una base de datos se suele utilizar en aquellos atributos sobre los que se realizan búsquedas frecuentes. El funcionamiento del índice es similar al de un libro: se guarda el elemento que se desea indexar y su posición en la base de datos (que, en el caso del libro, correspondería al número de páginas).

Los índices pueden crearse con una o más columnas y constituyen la base para poder realizar búsquedas rápidas al azar o un acceso ordenado a registros, evitando así una búsqueda secuencial que podría ser muy lenta en el caso de que se almacene mucha información. Los índices, generalmente, no se consideran parte de la base de datos, ya que son un complemento añadido.



Utilización de índices en las consultas para ser más efectivas.

La importancia de los **índices radica** en su capacidad para acelerar las operaciones de búsqueda y filtrado de datos en una base de datos. Al crear un índice en una columna, se crea una referencia adicional a los registros de la tabla basada en los valores de esa columna. Esto permite que el motor de la base de datos encuentre rápidamente los registros que cumplen con una condición de búsqueda, en lugar de tener que realizar una exploración completa de la tabla.

Existen varios tipos de índices que se utilizan en las bases de datos. Algunos de los más comunes son:

- **Índice de árbol B:** Este tipo de índice utiliza una estructura de árbol balanceado para organizar los datos y permitir una búsqueda eficiente. Es adecuado para columnas con valores no repetidos.

- **Índice de hash**: En este tipo de índice, se utiliza una función de hash para calcular una ubicación física directa donde se almacena el registro. Es eficiente para búsquedas exactas pero no es adecuado para rangos o consultas complejas.

- **Índice de bitmap:** Se utiliza para columnas con valores discretos y ofrece un mapa de bits que indica la presencia o ausencia de un valor en cada fila de la tabla.

- **Índice de texto completo:** Se utiliza para buscar texto dentro de columnas de tipo texto y permite búsquedas de palabras clave y expresiones regulares.

Las **claves alternativas y las claves candidatas** son candidatas naturales para ser convertidas en índices debido a su naturaleza única y su capacidad para identificar los registros de manera eficiente. Al crear índices en estas columnas, se mejora el rendimiento de las operaciones de búsqueda y consulta, ya que se puede acceder directamente a los registros mediante los valores de las claves.

Veamos a continuación cuáles son las **ventajas e inconvenientes** de trabajar con claves alternativas:

- **Ventajas:**
  - **Unicidad**: Las claves alternativas y candidatas suelen ser únicas, lo que significa que cada valor de la clave identifica de manera exclusiva un registro en la tabla. Esto permite un acceso más rápido a los registros mediante el índice, ya que no hay necesidad de buscar entre valores duplicados.
  - **Eficiencia en las consultas:** Al utilizar una clave alternativa o candidata como índice, las consultas que se basan en esos valores pueden ser más eficientes. El índice permite una búsqueda más rápida y directa de los registros que cumplen con los criterios de búsqueda, lo que mejora el rendimiento general de las consultas.
  - **Optimización de la velocidad de acceso:** Al tener índices en claves alternativas o candidatas, se puede optimizar la velocidad de acceso a los datos. Esto es especialmente beneficioso en tablas grandes con muchas filas, donde el índice facilita la localización rápida de los registros relevantes.

- **Desventajas**:
  - **Espacio de almacenamiento adicional:** La creación de índices en claves alternativas o candidatas puede ocupar espacio adicional en la base de datos. Cada índice creado requiere almacenamiento adicional para mantener la estructura de datos que permite una búsqueda rápida. Esto puede ser un factor importante a considerar en bases de datos con limitaciones de espacio.
  - **Mantenimiento de índices:** Los índices en claves alternativas o candidatas deben ser mantenidos y actualizados cada vez que se realizan modificaciones en los datos. Esto puede requerir recursos adicionales de tiempo y procesamiento, especialmente en tablas con cambios frecuentes.
  - **Costo de inserción y actualización**: La inserción y actualización de registros en tablas con índices puede ser más costosa en términos de rendimiento. Cada modificación en los datos puede requerir la actualización del índice correspondiente, lo que implica un costo adicional de procesamiento.

3.8 El valor NULL. Operar con el valor NULL

NULL es un valor fundamental en el modelo relacional, ya que permite manejar datos desconocidos o sin valor. Fue definido por el informático inglés Edgar Frank Codd y se utiliza para asignar un valor a un atributo que no contiene ninguno de los valores del dominio correspondiente.

En los sistemas de **gestión de bases de datos relacionales (DBMS)**, NULL se utiliza para representar información "desconocida" o "no aplicable". Para ilustrar esto, consideremos un ejemplo con una tabla de vehículos que almacena tanto motocicletas como automóviles.

Un **atributo** o campo en la tabla indica la posición del volante, para distinguir entre vehículos con el volante a la izquierda y aquellos con el volante a la derecha. Sin embargo, este atributo no tiene sentido en el caso de las motocicletas, por lo que se asigna un valor NULL al atributo correspondiente a las entidades de tipo motocicleta. El dominio de este atributo es "derecha" o "izquierda", por lo que si queremos asignar un valor dentro de ese dominio, la única opción es asignarle NULL. El valor NULL indica que el atributo no tiene ninguno de los valores posibles en el dominio (derecha o izquierda). De esta manera, el valor NULL proporciona información adicional.

Es importante destacar que no se pueden realizar operaciones **aritméticas** con el valor NULL (por ejemplo, 5 + NULL), pero sí es posible realizar operaciones lógicas. Estas operaciones lógicas solo tienen como resultado verdadero o falso. Sin embargo, cuando se involucra un valor nulo, se generan operaciones con resultados indeterminados, lo que da lugar al sistema de lógica ternaria. En este sistema, además de los valores verdadero y falso, se agrega el valor desconocido

p AND q		p		
		Verdadero	Falso	Desconocido
q	Verdadero	Verdadero	Falso	Desconocido
	Falso	Falso	Falso	Falso
	Desconocido	Desconocido	Falso	Desconocido

p OR q		p		
		Verdadero	Falso	Desconocido
q	Verdadero	Verdadero	Verdadero	Verdadero
	Falso	Verdadero	Falso	Desconocido
	Desconocido	Verdadero	Desconocido	Desconocido

p = q		p		
		Verdadero	Falso	Desconocido
q	Verdadero	Verdadero	Falso	Desconocido
	Falso	Falso	Verdadero	Desconocido
	Desconocido	Desconocido	Desconocido	Desconocido

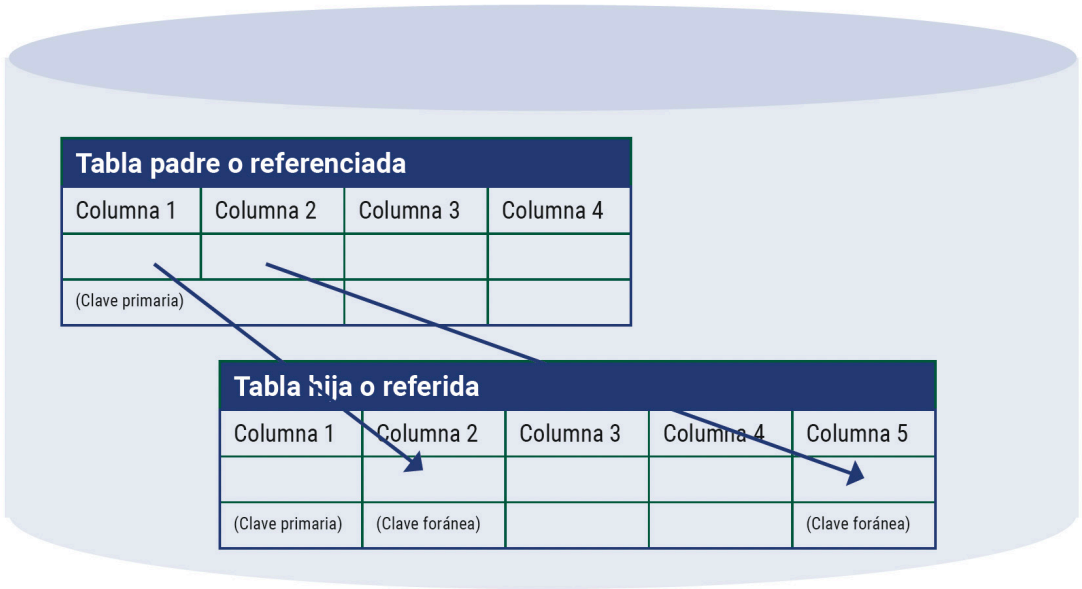
P	Not p
Verdadero	Falso
Falso	Verdadero
Desconocido	Desconocido

Tablas de resultados tras operar con el valor NULL.

PARA SABER MÁS: Existe una gran controversia con respecto a la lógica ternaria. Algunos expertos consideran que la existencia de un valor "desconocido" carece de la lógica necesaria para formar parte de operaciones lógicas.

3.9 Claves ajenas

La **clave ajena o foránea (foreign key)** es un conjunto de atributos en una tabla (tabla hija o referida) que se referencia con la clave primaria en otra tabla (tabla padre o referenciada) y determina la relación existente entre ambas tablas. Las claves foráneas no necesitan ser claves únicas en la tabla referida, pero sí, en cambio, en la tabla donde está referenciada.



Las claves foráneas relacionan una tabla referenciada con una tabla referida.

TABLA DE ALUMNOS		
DNI_ alumno	**Apellido**	**Nombre**
00000000 A 00000001 B 00000002 C	Sánchez Martínez Pérez	Pablo Sergio Álex

Tabla de alumnos.

TABLA DE NOTAS			
Codigo_asignatura	**Asignatura**	**DNI_ alumno**	**Nota**
M001 G002 I006	CMatemáticas Geografía Inglés	00000000 A 00000001 B 00000002 C	*C+* *B* *C-*

Tabla de notas.

A continuación, veremos cómo se relacionan dos tablas mediante las claves foráneas. Tenemos, por un lado, una tabla con un listado de alumnos identificados por el DNI (columna que reconoce a cada alumno de manera única) y, por otro lado, una tabla con una serie de notas identificadas por el DNI del alumno.

Se entiende que la Tabla de notas es foránea de la Tabla de alumnos, ya que es necesaria esta última para saber de quién son las notas. Además, si se cambiase el DNI de algún alumno, debería cambiarse en las dos tablas para que la referencia se mantuviese.

3.9.1 Relaciones y restricciones de claves ajenas

La relación entre dos tablas a través de una clave ajena implica que cada valor de la clave ajena en la tabla secundaria debe hacer referencia a un valor existente en la clave primaria correspondiente en la tabla principal. Esto establece una dependencia entre las dos tablas y asegura la consistencia de los datos.

Al definir una clave ajena, se pueden establecer restricciones para garantizar la integridad referencial. Estas restricciones incluyen:

- **Restricción de clave ajena:** Impide que se inserten registros en la tabla secundaria que no cumplan con la referencia a la clave primaria en la tabla principal.
- **Restricción de actualización en cascada:** Permite actualizar automáticamente los valores de la clave ajena en la tabla secundaria cuando se actualiza la clave primaria en la tabla principal.
- **Restricción de eliminación en cascada:** Permite eliminar automáticamente los registros en la tabla secundaria cuando se elimina el registro correspondiente en la tabla principal.

Estas restricciones aseguran la consistencia de los datos y evitan inconsistencias o referencias erróneas en las relaciones entre las tablas.

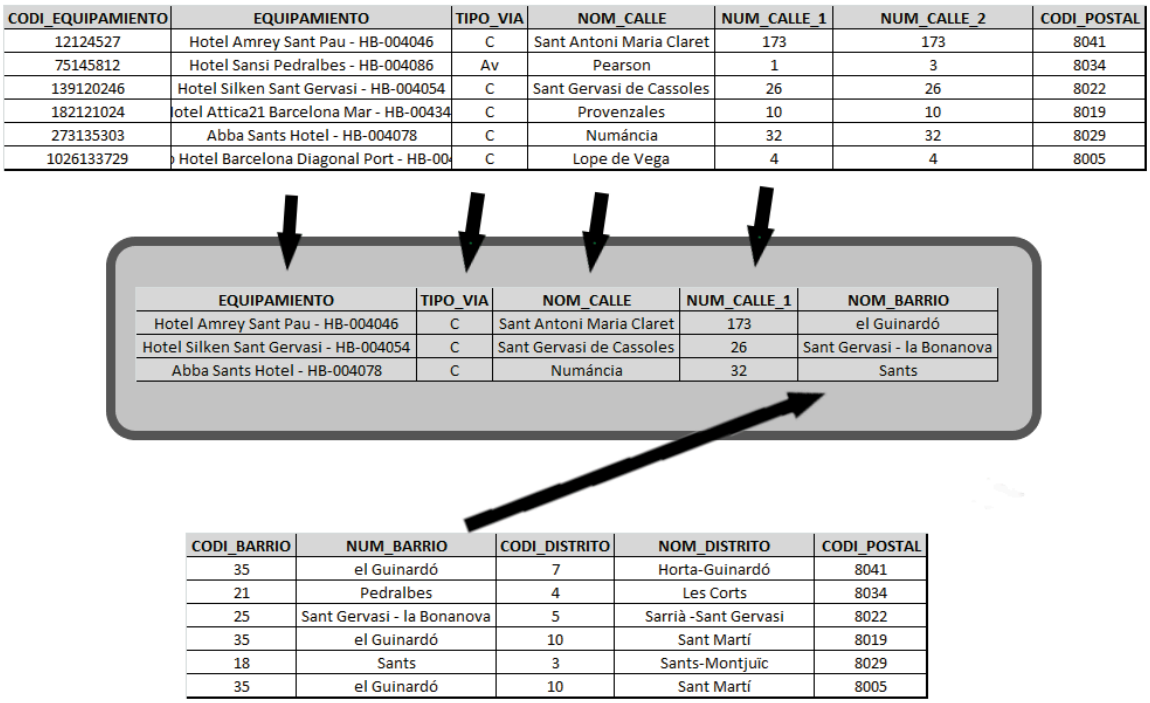
3.9.2. Mantenimiento y actualización de claves ajenas

El mantenimiento y la actualización de las claves ajenas son tareas importantes en la gestión de una base de datos. Algunas consideraciones clave incluyen:

- **Inserción de datos:** Cuando se inserta un nuevo registro en la tabla secundaria, se debe asegurar que los valores de la clave ajena correspondan a valores existentes en la tabla principal.
- **Actualización de datos:** Si se modifica el valor de la clave primaria en la tabla principal, se deben actualizar los valores de las claves ajenas en la tabla secundaria para mantener la integridad referencial.
- **Eliminación de datos:** Si se elimina un registro en la tabla principal, se deben tomar medidas para eliminar o actualizar los registros relacionados en la tabla secundaria, según la restricción de eliminación en cascada establecida.

El mantenimiento y la actualización adecuados de las claves ajenas son fundamentales para garantizar la consistencia y la integridad de los datos en una base de datos relacional. Esto implica seguir las reglas y restricciones definidas para las claves ajenas y realizar las operaciones de inserción, actualización y eliminación de datos de manera cuidadosa y precisa.

3.10 Vistas



Composición de una vista a partir de las columnas de diferentes tablas.

Una vista (view) puede definirse como una tabla lógica que permite acceder a los datos de otras tablas. Las tablas sobre las cuales se crea se denominan tablas base. El que una vista sea una tabla lógica significa que no contiene datos en sí misma, por lo que puede ser entendida como una tabla virtual que se nutre de datos de una o varias tablas. Una vez definida, puede ser utilizada en cualquier lugar (incluso en la definición de otra vista).

Al igual que sucede con una tabla, pueden insertarse, modificarse, borrarse y seleccionarse los datos en una vista; y en algunos casos existen restricciones en la actualización de sus datos.

En la anterior tabla podemos ver cómo a partir de diferentes columnas de dos tablas podemos crear una nueva vista que reúna todas estas columnas en una nueva tabla.

RECUERDA: Las vistas tienen la misma estructura que una tabla: filas y columnas.

3.10.1 Creación y gestión de vistas

La utilidad de las vistas radica en su capacidad para:

- **Simplificar la complejidad:** Las vistas permiten ocultar detalles complejos de la estructura y relaciones de las tablas subyacentes, proporcionando una interfaz más sencilla y manejable para los usuarios y aplicaciones.
- **Segmentar los datos:** Las vistas pueden mostrar una parte seleccionada de los datos almacenados en las tablas, lo que facilita el acceso a información relevante y reduce la carga de procesamiento.
- **Proporcionar seguridad:** Las vistas pueden utilizarse para restringir el acceso a ciertos datos sensibles. Se pueden definir permisos específicos para las vistas, lo que permite controlar qué usuarios o roles pueden ver y modificar los datos.
- **Simplificar consultas:** Las vistas pueden combinar datos de múltiples tablas en una sola vista, lo que ayuda a la creación de consultas más complejas y la generación de informes personalizados.
- **Mantener la consistencia:** Las vistas pueden aplicar reglas y cálculos predefinidos a los datos, asegurando así que los resultados sean consistentes y estandarizados en todas las consultas.

3.10.2 Creación y gestión de vistas

Las vistas se crean mediante sentencias SQL específicas que definen los campos, las tablas y las condiciones de filtrado o unión necesarios para construir la vista. Una vez creada, la vista se almacena en la base de datos y se puede acceder a ella como si fuera una tabla real.

La gestión de vistas implica tareas como:

- **Creación de vistas:** Se define la estructura y los criterios de selección de datos para crear la vista.
- **Modificación de vistas:** Se pueden realizar cambios en la definición de la vista, como agregar o eliminar campos o modificar las condiciones de filtrado.
- **Eliminación de vistas:** Las vistas se pueden eliminar cuando ya no son necesarias.
- **Actualización de vistas:** En algunos casos, se pueden permitir operaciones de actualización en las vistas, lo que permite modificar los datos subyacentes a través de la vista.

3.10.3 Beneficios y consideraciones al utilizar vistas

- **Simplificación de la complejidad:** Permiten a los usuarios trabajar con una representación más simple y comprensible de los datos, sin tener que preocuparse por la estructura y las relaciones complejas de las tablas subyacentes.
- **Seguridad mejorada:** Pueden utilizarse para limitar el acceso a ciertos datos sensibles, asegurando que solo los usuarios autorizados puedan ver o modificar esos datos.
- **Mejora del rendimiento:** Al utilizar vistas, se pueden realizar consultas más eficientes y optimizadas, ya que se pueden reducir la cantidad de datos accedidos y procesados.
- **Flexibilidad en la presentación de datos:** Las vistas permiten personalizar la forma en que se presentan los datos, lo que facilita la generación de informes y la visualización de información relevante para cada usuario o aplicación.
- **Mantenimiento centralizado:**Se puede centralizar la lógica de consulta y los cálculos en un solo lugar, lo que facilita la actualización y el mantenimiento de la base de datos.

3.11 Usuarios, privilegios y roles

Para conservar la integridad de los datos y sus estructuras es imprescindible que sólo algunos usuarios puedan realizar determinadas tareas. Conceptos como usuarios y privilegios están íntimamente relacionados, ya que no puede crearse un usuario sin asignarle al mismo tiempo privilegios y restricciones.

3.11.1 Administración de usuarios en una base de datos

La administración de usuarios en una base de datos implica la creación, modificación y eliminación de cuentas de usuario que pueden acceder y utilizar la base de datos. Los usuarios se crean para permitir el acceso controlado a la información almacenada y para asignar responsabilidades específicas a diferentes personas o aplicaciones.

Al administrar usuarios en una base de datos, se realizan tareas como:

- **Creación de usuarios:** Se crean cuentas de usuario con identificadores únicos y se les asigna una contraseña para acceder a la base de datos.
- **Modificación de usuarios:** Se pueden realizar cambios en las propiedades de los usuarios existentes, como su nombre, contraseña o privilegios.
- **Eliminación de usuarios:** Cuando un usuario ya no necesita acceso a la base de datos, su cuenta se puede eliminar para revocar sus privilegios de acceso.

3.11.2 Asignación de privilegios y permisos

Un privilegio es un permiso dado a un usuario dentro de la base de datos para ejecutar determinadas operaciones o acceder a determinados objetos.

Éste puede clasificarse en dos categorías:

- **Privilegios de sistema.** Permiten crear una acción particular dentro del sistema o una acción particular sobre un tipo determinado de objeto (como nuevas tablas) y adjudicarles propiedades, niveles de seguridad, etc. La mayoría están disponibles sólo para administradores y desarrolladores de aplicaciones.
- **Privilegios de objetos.** Permiten crear acciones sobre un objeto específico, como borrar filas de una tabla específica. Estos privilegios son asignados normalmente a usuarios finales.

Al asignar privilegios y permisos a los usuarios, se pueden especificar acciones como:

- **Lectura**: Permite a los usuarios ver los datos en una tabla o vista.
- **Escritura**: Permite a los usuarios insertar, actualizar o eliminar datos en una tabla o vista.
- **Ejecución**: Permite a los usuarios ejecutar procedimientos almacenados o funciones.
- **Administración**: Permite a los usuarios realizar tareas administrativas, como crear tablas o realizar copias de seguridad.

3.11.3 Uso de roles para gestionar el acceso

En cuanto a un rol, éste no es más que un conjunto de privilegios del sistema y de objetos agrupados con un determinado nombre. Son un mecanismo para simplificar y administrar la asignación de privilegios y permisos en una base de datos. Un rol es un conjunto predefinido de privilegios y permisos que se asigna a un grupo de usuarios con características y responsabilidades similares.

El usuario puede recibir un privilegio de dos formas distintas:

- Asignado a los usuarios explícitamente.
- Asignado a roles (grupo de privilegios), que después pueden ser asignados a uno o más usuarios.

3.12 Lenguaje de descripción de datos (DDL)

El **DDL (Data Definition Language)** o Lenguaje de Definición de Datos es un lenguaje proporcionado por el sistema de gestión de la base de datos que permite a los usuarios de ésta llevar a cabo las tareas de definición o modificación de la estructura de los objetos que contiene.

El DDL permite crear, modificar y eliminar tablas, vistas, índices, restricciones y otros elementos de la base de datos.

Las principales funciones del lenguaje DDL son:

- **Creación de objetos de base de datos:** Permite crear tablas, vistas, índices, restricciones y otros objetos en la base de datos. Se especifica la estructura, los atributos y las propiedades de cada objeto.
- **Modificación de objetos de base de datos:** Permite realizar cambios en la estructura de los objetos existentes. Se pueden agregar, modificar o eliminar columnas, cambiar las propiedades de los objetos y realizar otras modificaciones según sea necesario.
- **Eliminación de objetos de base de datos**: Permite eliminar objetos de la base de datos cuando ya no son necesarios. Esto implica eliminar tablas, vistas, índices u otros elementos de la base de datos.

A diferencia de muchos lenguajes DDL, el **SQL (Structured Query Language)** utiliza una serie de verbos imperativos, llamados **sentencias**, cuyo efecto es modificar el esquema de la base de datos, añadir, cambiar o eliminar las definiciones de tablas y otros objetos.

Estas declaraciones pueden mezclarse libremente con otras sentencias SQL, por lo que el DDL no es realmente un lenguaje independiente en SQL.

El lenguaje de programación SQL, el más difundido entre los gestores de bases de datos, admite las siguientes sentencias con sus respectivas cláusulas de definición de datos:

- **CREATE**. Crea una tabla en la que definiremos los atributos que tendrá así como el formato y las características de clave primaria y foránea.

```
CREATE TABLE nombre_tabla (
  columna1 tipo_de_datos_para_columna_1,
  columna2 tipo_de_datos_para_columna_2,
  ... );
```

- **ALTER**. Modifica la estructura de una tabla ya creada. Podemos añadir o modificar columnas, e incluso cambiar sus características.

```
ALTER TABLE table_name
[ADD column_name datatype] *crea una nueva columna dentro
de la tabla
[DROP COLUMN column_name] *elimina una columna
[ALTER COLUMN column_name datatype] *cambia las propiedades
de la columna
```

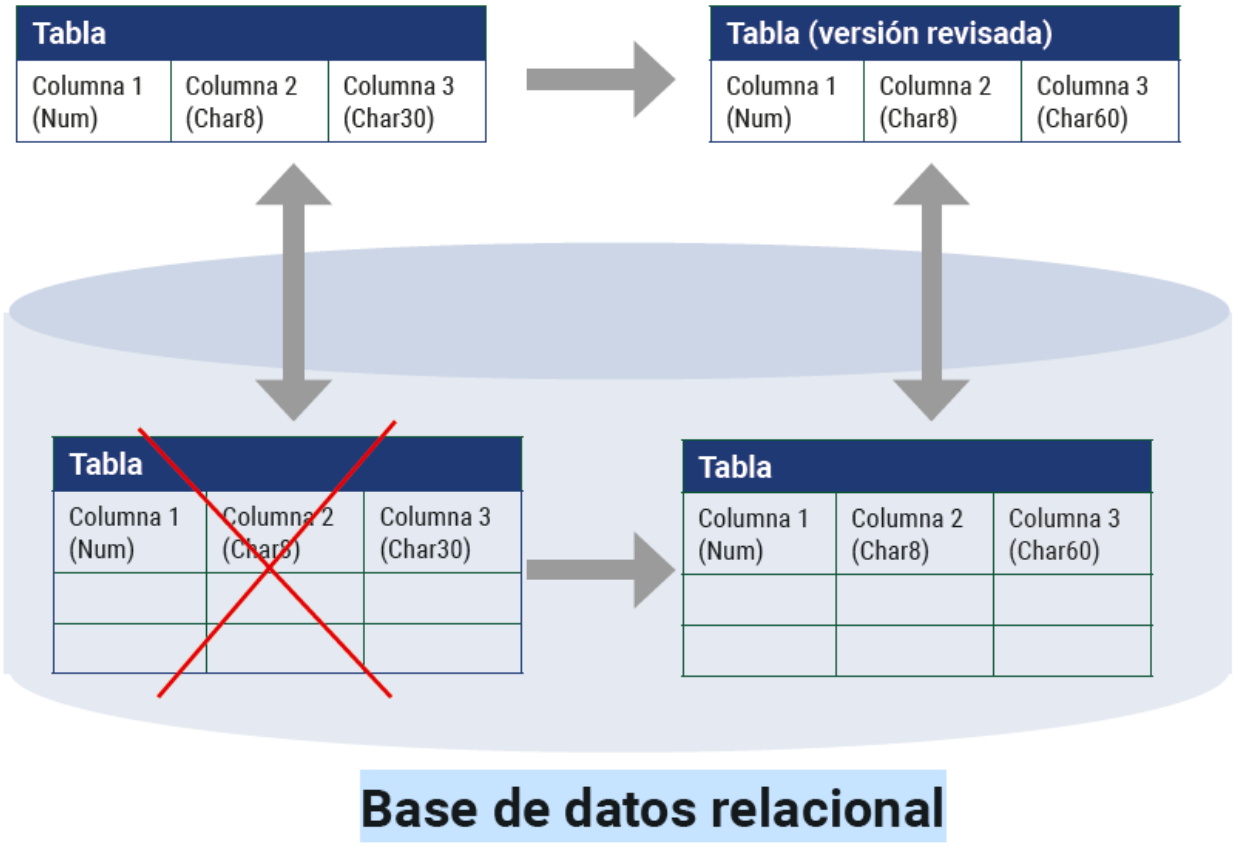
- **DROP**. Elimina una tabla creada.

```
DROP TABLE table_name
```

- **TRUNCATE.** Vacía todo el contenido de una tabla. Esta sentencia parece una DML (Data Manipulation Language) o Lenguaje de Manipulación de Datos; pero a diferencia de éste, TRUNCATE borra la tabla y la vuelve a crear sin ejecutar ninguna transacción.

```
TRUNCATE TABLE table_name
```

PARA SABER MÁS: Un procedimiento almacenado (stored procedure) es un programa guardado en una base de datos. Se ejecuta directamente en el motor de bases de datos y sólo necesita enviar sus resultados al usuario, por lo que elimina la sobrecarga de comunicar grandes cantidades de datos salientes y entrantes. Los procedimientos almacenados y utilizados para la validación de datos se conocen, comúnmente, con el nombre de "disparadores" (triggers).



El lenguaje DDL permite modificar la estructura de la base de datos.

Los tipos de objetos a los que pueden aplicarse dichas sentencias dependen del gestor de base de datos que esté siendo utilizado. La mayoría permite su aplicación en tablas, vistas, índices, usuarios, sinónimos, procedimientos almacenados (stored procedure), consultas almacenadas (triggers) y bases de datos.

El uso de comandos DDL en la administración de la estructura de la base de datos es fundamental para garantizar la integridad y consistencia de los datos almacenados. A través de estos comandos, se pueden crear, modificar y eliminar objetos de la base de datos, como tablas, vistas, índices y restricciones.

Con los comandos DDL, es posible crear la estructura inicial de la base de datos y adaptarla a medida que cambian los requisitos. También se pueden establecer restricciones y reglas para asegurar la validez de los datos ingresados. Además, los comandos DDL permiten crear vistas personalizadas que simplifican el acceso a los datos y mejorar el rendimiento mediante la creación de índices.

3. 13 Lenguaje de control de datos (DCL)

El DCL (Data Control Language) o Lenguaje de Control de Datos es un lenguaje proporcionado por el sistema de gestión de base de datos que permite al administrador controlar el acceso a los datos contenidos en la base de datos y gestionar la seguridad de la misma.

El lenguaje DCL se utiliza para establecer políticas de acceso y controlar las operaciones que los usuarios pueden realizar en la base de datos. Algunas de las características principales del lenguaje DCL son:

- **Gestión de permisos:** Permite otorgar o revocar privilegios y permisos a los usuarios sobre los objetos de la base de datos, como tablas, vistas, procedimientos almacenados, entre otros. Los comandos DCL, como GRANT y REVOKE, se utilizan para asignar y retirar permisos específicos a los usuarios, como SELECT, INSERT, UPDATE o DELETE, según las necesidades de seguridad y privacidad de la base de datos.
- **Control de acceso:** El lenguaje DCL también se utiliza para definir políticas de acceso y controlar quién puede acceder a la base de datos y a qué objetos. Mediante comandos como CREATE USER y DROP USER, los administradores pueden crear y eliminar cuentas de usuario, establecer contraseñas y asignar roles.

El SQL, a diferencia de muchos lenguajes DCL, incluye una serie de sentencias que permiten al administrador asignar a los usuarios permisos para realizar tareas en la base de datos. El DCL no es realmente un lenguaje independiente en SQL; no debemos olvidar este aspecto a la hora de trabajar en un entorno multiusuario, en el que se debe tener en cuenta la protección de los datos, la seguridad de las tablas y el establecimiento de restricciones en el acceso, así como los elementos para coordinar y compartir los datos por parte de usuarios concurrentes, y asegurarnos de que no interfieren unos con otros.

La existencia de estas sentencias del Lenguaje de Control de Datos dependerá de la implementación del estándar SQL que lleve a cabo el gestor de base de datos que está siendo utilizado.

Veamos a continuación algunos ejemplos de sentencias con sus respectivas cláusulas incluidas en el DCL:

- **GRANT**. Permite asignar privilegios o permisos a uno o varios usuarios, así como roles para realizar tareas determinadas, como acceder a los datos o modificarlos.

```
GRANT nombre_del_privilegio
ON objeto
TO {usuario |PUBLIC |rol}
[WITH GRANT OPTION];
```

- **REVOKE**. Permite eliminar o revocar los privilegios que previamente se han concedido con GRANT.

```
REVOKE nombre_del_privilegio
ON objeto
FROM { usuario |PUBLIC |role_name}
```

- **CREATE ROL.** Es una colección de privilegios y derechos de acceso que pueden ser aplicados a un gran número de usuarios al mismo tiempo.

```
CREATE ROLE test; -- Crea un rol con el nombre "test"
GRANT CREATE TABLE TO test; -- Otorga permisos al rol "test" para crear tablas
GRANT test TO usuario1; -- Asigna el rol "test" a un usuario
```

- **CREATE USER.** Crea un usuario al que se le aplicarán unos permisos determinados.

```
CREATE USER usuario [IDENTIFIED BY 'contraseña']
```

El lenguaje DCL también se utiliza para gestionar la seguridad de la base de datos y llevar a cabo la auditoría de las actividades realizadas por los usuarios. Algunas acciones que se pueden realizar con los comandos DCL son:

- **Creación de roles:** Los roles son conjuntos de privilegios que se pueden asignar a un grupo de usuarios. Los comandos DCL permiten crear roles y asignarles permisos específicos, lo que facilita la gestión de la seguridad y simplifica el proceso de asignación de permisos a múltiples usuarios.
- **Gestión de perfiles:** Los perfiles se utilizan para establecer límites y restricciones en el uso de recursos de la base de datos por parte de los usuarios. Con los comandos DCL, es posible crear y administrar perfiles para controlar el consumo de recursos, como espacio en disco o tiempo de CPU, y asignarlos a los usuarios correspondientes.
- **Auditoría de la base de datos:** Los comandos DCL permiten habilitar la auditoría de la base de datos, lo que implica el seguimiento y registro de las actividades realizadas por los usuarios. Esto incluye la generación de registros de auditoría que registran eventos como inicios de sesión, consultas ejecutadas, modificaciones de datos, entre otros. La auditoría proporciona un mecanismo de control y seguimiento para garantizar la integridad y seguridad de la base de datos.

4. Resumen

Las bases de datos relacionales se caracterizan por la simplicidad de su diseño mediante la representación de los datos en tablas relacionadas entre ellas. Estas relaciones proporcionan al usuario pistas para poder contextualizar los datos en la vida real en vez de ser mera información digitalizada.

En las tablas pueden existir claves primarias que identifican de manera inequívoca cada uno de los registros almacenados. Esto es especialmente útil para obtener datos específicos, ya que al ser identifi cados con un elemento único favorece las operaciones de búsqueda y ordenación por parte del servidor. Las claves ajenas son aquellas enlazadas con claves primarias de otras tablas; de esta manera se consigue establecer las relaciones entre las tablas.

Las tablas que no disponen de claves primarias tienen la posibilidad de optimizar las búsquedas de registros mediante la incorporación de índices en los registros. Los índices suelen ser asignados de manera automática y correlativa por el sistema, al insertar un nuevo registro, lo que evita que existan índices repetidos.

Las vistas son estructuras conceptuales creadas por los usuarios para reunir en una sola tabla los datos de diferentes objetos con el fi n de obtener la información que se ajuste a sus necesidades. De esta manera los usuarios pueden crear tablas personalizadas en las que podrán aplicar consultas tal y como lo harían con una tabla normal.

El valor NULL es un concepto utilizado en las bases de datos para representar aquellos registros en los que no se ha introducido ningún valor. No es considerado un valor real, y a pesar de que no pueden realizarse operaciones aritméticas sobre el valor NULL, los sistemas de bases de datos y la programación en general permite realizar operaciones lógicas y de comparación con este valor, siguiendo la denominada lógica ternaria.

El acceso a una base de datos está restringido a los usuarios registrados en ella, y cada usuario puede tener diferentes privilegios sobre los objetos, sean tablas o vistas. De esta manera pueden definirse usuarios que sólo podrán realizar consultas, o usuarios que pueden modificar datos. Los roles son definidos para agrupar diferentes usuarios con los mismos permisos.

El Lenguaje de Descripción de Datos (DDL) engloba todas aquellas consultas realizadas a la base de datos por aquellos usuarios que tienen como objetivo definir o modificar las estructuras de los objetos de la base de datos. Dentro de esta categoría se hallan las instrucciones para crear tablas y vistas, defi nir las columnas, vaciar el contenido de una tabla, modificar nombres y parámetros, etc.

En cambio, el Lenguaje de Control de Datos (DCL) se encarga de crear y modificar los permisos de los usuarios sobre los objetos de la base de datos.

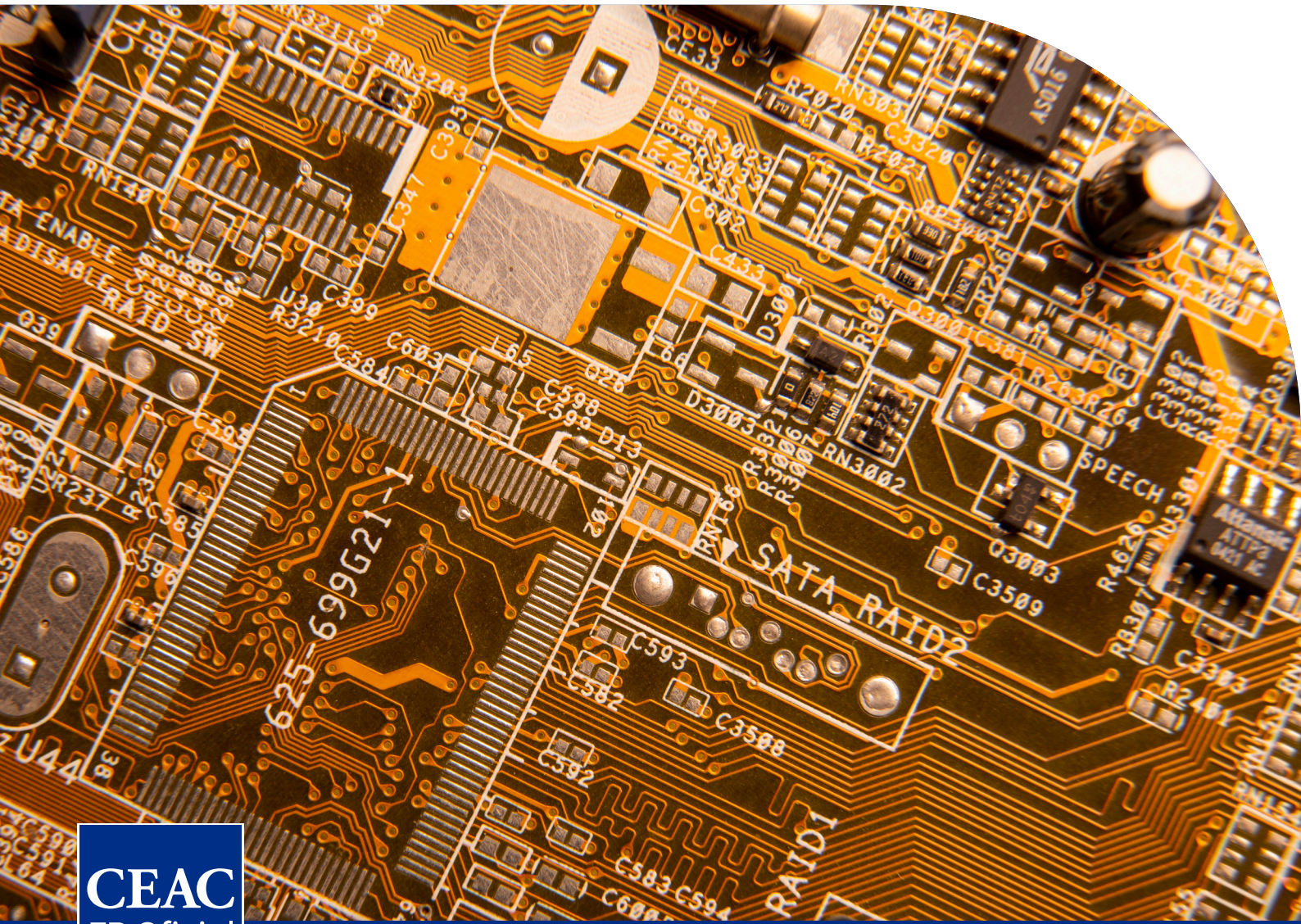
Desarrollo de aplicaciones multiplataforma

MÓDULO:

Base de datos

UNIDAD:

Realización de consultas



CEAC
FP Oficial

MÓDULO:

Base de datos

UNIDAD:

Realización de consultas

CEAC FP Oficial

ÍNDICE

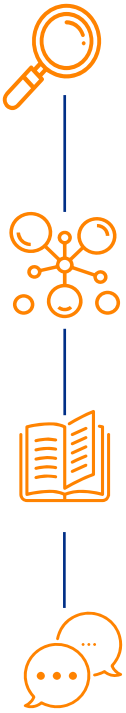
1. Introducción

Una vez hemos aprendido cómo se estructuran las bases de datos, analizaremos una parte fundamental del módulo: la realización de consultas. Las consultas son el modo que tenemos de obtener o insertar la información en las bases de datos para poder utilizarla posteriormente. A lo largo de este capítulo, estudiaremos los diferentes tipos de consultas que existen, y conoceremos cuál es su funcionalidad y cuál su estructura.

A pesar de que las diferentes bases de datos pueden tener algunas diferencias en la construcción de las consultas, la mayoría de ellas sigue unas pautas determinadas y se rige por las mismas normas con el fin de conseguir una mayor estandarización entre ellas.

2. Objetivos pedagógicos

Los objetivos de aprendizaje que se pretenden alcanzar con esta unidad son:



- **Entender el propósito y la importancia de las consultas.**
- **Aprender los diferentes tipos de consultas.**
- **Familiarizarse con la estructura de las consultas.**
- **Reconocer la estandarización en la construcción de consultas.**

3. Realización de consultas

3.1 Herramientas gráficas proporcionadas por el sistema gestor para la realización de consultas

Las herramientas gráficas de consulta son interfaces visuales que facilitan la interacción y manipulación de bases de datos mediante una representación gráfica de los elementos de la base de datos. Estas herramientas permiten a los usuarios realizar consultas y obtener resultados de manera intuitiva, sin necesidad de escribir consultas SQL manualmente.

Estas herramientas proporcionan una interfaz gráfica de usuario (GUI) que muestra de manera visual las tablas, columnas y relaciones de la base de datos. Los usuarios pueden interactuar con la base de datos seleccionando las tablas, estableciendo criterios de búsqueda, definiendo columnas a mostrar y aplicando filtros y ordenamientos de forma visual.

3.1.1 Funcionalidades y características de las herramientas gráficas

Las herramientas gráficas de consulta ofrecen diversas funcionalidades y características para facilitar el proceso de consulta de datos. Algunas de las más comunes son:

a. Diseño visual de consultas: Estas herramientas permiten arrastrar y soltar tablas y columnas en una interfaz gráfica para diseñar visualmente las consultas. Los usuarios pueden seleccionar las tablas y columnas deseadas, definir criterios de búsqueda y establecer relaciones entre las tablas de forma intuitiva.

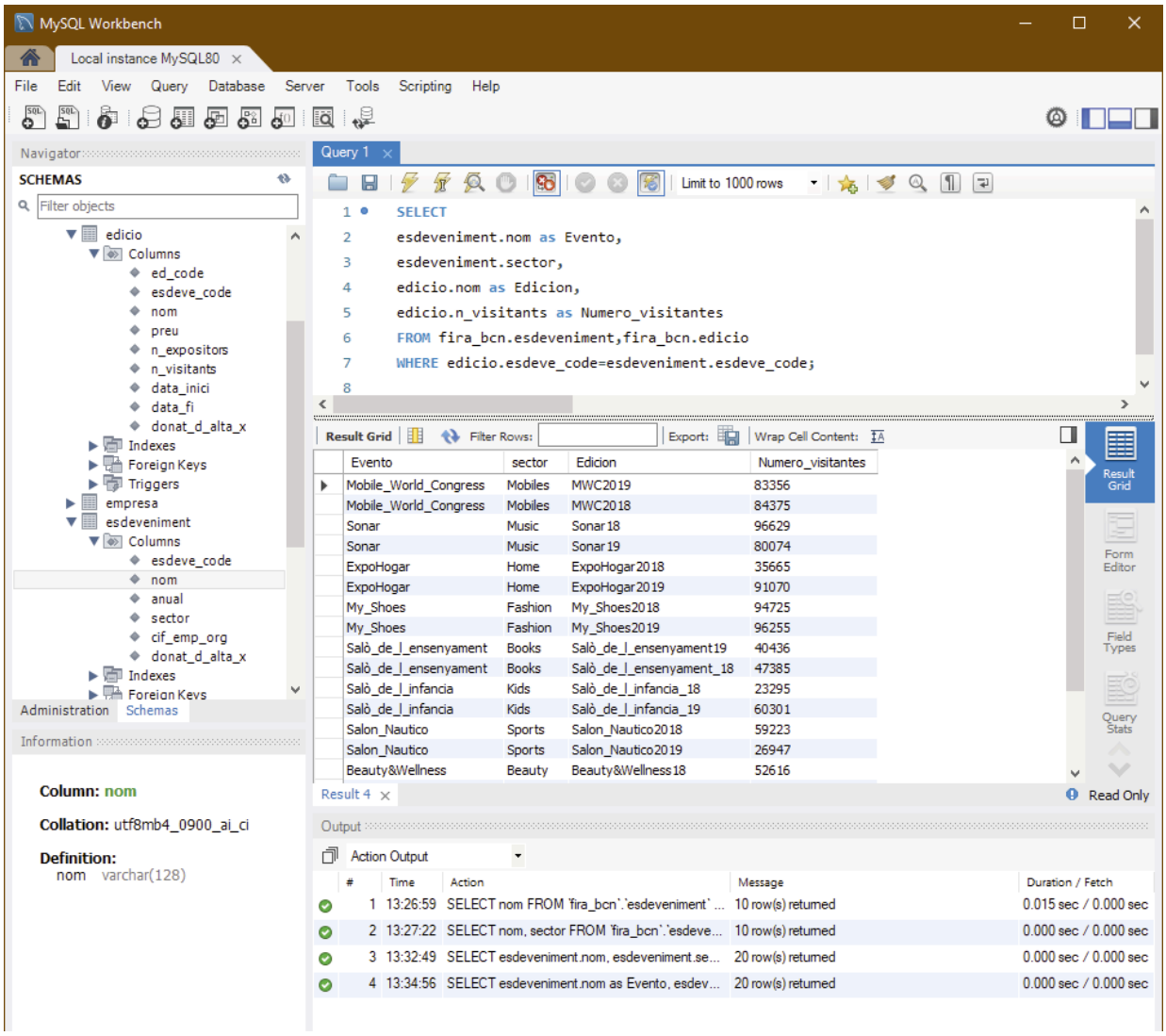
b. Generación automática de consultas: Al seleccionar tablas y columnas, las herramientas generan automáticamente el código SQL correspondiente a la consulta diseñada. Esto evita la necesidad de escribir consultas SQL manualmente y facilita el proceso de construcción de consultas complejas.

c. Visualización de resultados: Las herramientas gráficas muestran los resultados de las consultas de manera visual, generalmente en forma de tablas o informes. Los usuarios pueden explorar y analizar los datos de manera interactiva, realizar operaciones de filtrado, ordenamiento y agrupamiento en los resultados obtenidos.

d. Creación y edición de consultas guardadas: Estas herramientas permiten a los usuarios guardar consultas realizadas previamente para su posterior uso. También brindan la capacidad de editar y modificar consultas existentes, lo que facilita el proceso de refinamiento de las consultas a medida que se desarrollan nuevas necesidades.

e. Soporte para múltiples bases de datos: Las herramientas gráficas suelen ser compatibles con diversos sistemas gestores de bases de datos, lo que permite a los usuarios conectarse y trabajar con diferentes bases de datos desde una sola interfaz.

f. Interfaz intuitiva y amigable: Estas herramientas están diseñadas para ser fáciles de usar y comprender, incluso por usuarios no expertos en SQL. Suelen contar con elementos gráficos, menús desplegables, asistentes y otros elementos visuales.



Ejemplo de interfaz gráfica en SGBD.

Es recomendable probar los diferentes SGBD que existen en el mercado para familiarizarnos con ellos. Durante el curso, utilizaremos el Workbench de MySQL, un software libre de gestión de bases de datos disponible parabtodas las plataformas. Se puede descargar desde la dirección: http://www.mysql.com/.

RECUERDA: Para seleccionar todas las columnas en una consulta SELECT, es necesario utilizar el símbolo asterisco (*). La consulta SELECT * FROM [tabla] devolverá todos los valores que existen en la tabla.

3.2 La sentencia SELECT

La sentencia SELECT es una parte fundamental del lenguaje SQL (Structured Query Language) utilizada para recuperar datos de una base de datos. Su propósito principal es seleccionar columnas específicas de una o varias tablas y filtrar los resultados según ciertos criterios.

En la misma consulta podremos obtener datos de una o más tablas, así como ordenarlos o discriminarlos según nuestros criterios. A lo largo de este capítulo conoceremos la mayoría de configuraciones que permite esta sentencia:

Veamos a continuación la sintaxis básica de una consulta SELECT:

- **SELECT (seleccionar).** Definiremos las columnas de las que queremos obtener su valor. En la misma consulta, seleccionaremos varias columnas de diferentes tablas, siempre y cuando especifiquemos las relaciones entre tabla y columna, ya que las columnas de diferentes tablas pueden tener el mismo nombre.

```
SELECT [nombre_columna]
FROM [tabla]
WHERE [condicion]
ORDER BY [Asc/Desc]
```

- **FROM (desde).** Numeraremos las tablas a las que queramos tener acceso. Podremos elegir tantas tablas como queramos, aunque es recomendable acceder sólo a aquellas tablas que contengan los datos que necesitamos, de lo contrario cargaremos el sistema de forma innecesaria.

  En esta estructura, se especifican las columnas que se desean seleccionar separadas por comas y se indica la tabla de donde se obtendrán los datos.

  En la sentencia SELECT, se pueden seleccionar columnas específicas de una tabla o incluso combinar columnas de varias tablas en una sola consulta. Esto permite personalizar los resultados de acuerdo con los requisitos de la consulta.

  Las herramientas gráficas proporcionan interfaces intuitivas para seleccionar las columnas deseadas. Los usuarios pueden explorar la estructura de las tablas disponibles en la base de datos y seleccionar las columnas de interés mediante simples clics o arrastrando y soltando.

  Además, las herramientas gráficas permiten realizar selecciones complejas, como la agregación de columnas, la aplicación de funciones de manipulación de datos y la inclusión de expresiones personalizadas en la consulta.

- **WHERE (donde).** Se utiliza para filtrar los resultados de la consulta según ciertas condiciones. Permite especificar criterios de búsqueda para obtener solo los registros que cumplan con dichos criterios.

  Esta cláusula es opcional y, en caso de no ponerla en la consulta, nos devolverá todas las filas de la columna que hayamos seleccionado en el SELECT.ORDER BY (ordenar por). La consulta SELECT nos permite, además, ordenar alfabética o numéricamente los resultados. Un ejemplo sería la ordenación por apellido de una lista de alumnos. Veamos a continuación algunos ejemplos de esta sentencia utilizando la siguiente tabla.

P_Id	Apellido	Nombre	Ciudad
1	Sánchez	José	Barcelona
2	Alonso	Enrique	Málaga
3	Gutiérrez	Pablo	Madrid
4	Martínez	Eduardo	Madrid

- **ORDER BY (ordenar por).** La cláusula ORDER BY se utiliza para ordenar los resultados de la consulta en un orden específico, ya sea ascendente (ASC) o descendente (DESC), según los valores de una o más columnas.

Con esta tabla en nuestra base de datos:

1. Seleccionaremos todas las personas de las que tenemos constancia y las ordenaremos según su apellido:

```
SELECT * FROM Personas
ORDER BY Apellido
```

P_Id	Apellido	Nombre	Ciudad
1	Alonso	Enrique	Málaga
2	Gutiérrez	Pablo	Madrid
3	Martínez	Eduardo	Madrid
4	Sánchez	José	Barcelona

2. Seleccionaremos todas las personas de las que tenemos constancia que viven en la ciudad de Barcelona:

```
SELECT * FROM Personas
WHERE Ciudad=Barcelona
```

Y como resultado obtendremos:

P_Id	Apellido	Nombre	Ciudad
1	Sánchez	José	Barcelona

3.3 Consultas calculadas. Sinónimos

A continuación, conoceremos qué son las consultas calculadas y los sinónimos, y cómo inciden éstos en los cálculos matemáticos y en la construcción de sentencias.

3.3.1 Consultas calculadas

La mayoría de bases de datos nos permite insertar cálculos matemáticos en las consultas para obtener los datos de una operación entre dos o más campos. Imaginémonos que una hoja de cálculo es una base de datos. Para poder obtener el valor deseado realizaremos una operación aritmética sobre dos o más campos. Imaginemos que deseamos obtener una consulta en la que, además de la información de la base de datos, deseamos crear una nueva columna con el resultado de multiplicar dos valores. La sentencia que produciría como resultado:

```
SELECT Codigo_Edicion, Edicion, Numero_visitantes,(Precio_entrada * Numero_
visitantes) as Ingresos_en_€ FROM Ediciones
```

Codigo_Edicion	Edicion	Numero_visitantes	Ingresos_en_€
1	MWC2019	83356	3667664
2	Salò_de_l_infancia_18	23295	815325
3	EcoSalut_19	10343	186174
4	Salò_de_l_ensenyament19	40436	1010900
5	ExpoHogar2018	35665	1105615
6	Salon_Nautico2018	59223	1658244
7	My_Shoes2018	94725	3031200

Ejemplo de consulta calculada.

En esta consulta, estás seleccionando las columnas "Codigo_Edicion", "Edicion" y "Numero_visitantes" de la tabla "Ediciones". Además, se ha agregado una columna calculada llamada "Ingresos_en_€" que se obtiene multiplicando el "Precio_entrada" por el "Numero_visitantes".

3.3.2 Sinónimos

Los **sinónimos** son un recurso para facilitar la construcción de sentencias, son alias o nombres alternativos que se pueden asignar a tablas, columnas u otros objetos de la base de datos.

Si queremos efectuar operaciones entre diferentes tablas, debemos concatenar la tabla con sus respectivos campos, utilizando el punto (.). Estas concatenaciones pueden ser muy largas, pero con los sinónimos podemos definirlas con una sola palabra para facilitar la programación y comprensión de la consulta. Para crear un sinónimo, utilizamos la cláusula **AS** (como).

En las herramientas de gestión de bases de datos, se proporcionan funcionalidades para definir y utilizar sinónimos en las consultas. Los usuarios pueden asignar un alias a una tabla, columna u otro objeto utilizando una sintaxis específica. Por ejemplo, se puede asignar el sinónimo "cliente" a la tabla "TablaClientes".

Si en las dos tablas de la página siguiente (Ciudades y Personas) deseamos obtener el nombre de la ciudad y de la persona con un ID determinado podemos utilizar los sinónimos para definir los nombres como Ciudad y Persona. La sentencia Transact SQL que ilustra esta explicación es:

```
SELECT ciudad.nombre as cityName, persona.nombre as personName
```

3.4 Selección y ordenación de registros

La cláusula WHERE en una consulta SQL permite filtrar los registros de una tabla basándose en condiciones específicas. Con esta cláusula, puedes especificar una condición lógica que los registros deben cumplir para ser incluidos en el resultado de la consulta.

Por otro lado, la cláusula ORDER BY se utiliza para especificar el orden en el que se mostrarán los registros en el resultado de una consulta. Puedes ordenar los registros según uno o más campos de la tabla, en orden ascendente o descendente. Vemos a continuación un ejemplo de esta sentencia partiendo de que la tabla pertenece a nuestra base de datos:

P_Id	Apellido	Nombre	Ciudad
1	Sánchez	José	Barcelona
2	Alonso	Enrique	Málaga
3	Gutiérrez	Pablo	Madrid
4	Martínez	Eduardo	Madrid
5	Martínez	Alicia	Madrid

En este caso, la tabla tiene atributos llamados P_Id, Apellido, Nombre y Ciudad.

Para ordenar los resultados de la consulta por el atributo Id, se utiliza la cláusula ORDER BY seguida del nombre del atributo. Esto permite especificar el orden en el que se mostrarán los registros en el resultado de la consulta.

Por ejemplo, seleccionaremos todas las personas de las que tenemos constancia que viven en las ciudades que empiezan por la letra M y las ordenaremos por el apellido:

```
SELECT * FROM Personas

WHERE Ciudad LIKE 'M%'

ORDER BY Apellido;
```

P_Id	Apellido	Nombre	Ciudad
2	Alonso	Enrique	Málaga
3	Gutiérrez	Pablo	Madrid
4	Martínez	Eduardo	Madrid
5	Martínez	Alicia	Madrid

En este caso, la consulta seleccionará todas las columnas (representadas por el asterisco *) de la tabla Personas donde el valor de la columna Ciudad comience con la letra "M" seguida de cualquier otro carácter. La cláusula WHERE con la condición "Ciudad LIKE 'M%'" filtra los registros según este criterio.

Luego, la cláusula ORDER BY se utiliza para ordenar los resultados por el atributo Apellido en orden ascendente.

Y si, por ejemplo, seleccionáramos todas las personas y las ordenáramos por el apellido y el nombre, la consulta quedarías así:

```
SELECT * FROM Personas

ORDER BY Apellido, nombre
```

P_Id	Apellido	Nombre	Ciudad
2	Alonso	Enrique	Málaga
3	Gutiérrez	Pablo	Madrid
4	Martínez	Eduardo	Madrid
5	Martínez	Alicia	Madrid
1	Sánchez	José	Barcelona

Esta consulta seleccionará todas las columnas de la tabla Personas y las ordenará primero por el atributo Apellido y luego por el atributo Nombre en orden ascendente.

3.5 Operadores

En las consultas podremos utilizar operadores lógicos para filtrar los resultados de las búsquedas. Se colocarán junto a la cláusula **WHERE** y servirán para realizar operaciones entre dos columnas o entre una columna y un valor dado. Estos operadores son comunes a todos los sistemas de bases de datos gracias a la supervisión del **Instituto Nacional Estadounidense de Estándares** (ANSI, American National Standards Institute) a la hora de crear un estándar. Pese a ello, algunos SGBD utilizan ciertas peculiaridades para construir consultas, como pueden ser, saltos de línea o la incorporación del ";" al final de cada sentencia.

3.5.1 Operadores de comparación

Los operadores de comparación se utilizan para comparar valores en una consulta y devolver resultados basados en la condición evaluada. Algunos de los operadores de comparación comunes son:

	Apellido	Nombre	Ciudad
==	Igual	$a == $b	$a es igual a $b
!=	Distinto	$a != $b	$a es distinto a $b
<	Menor que	$a < $b	$a es menor que $b
>	Mayor que	$a > $b	$a es mayor que $b
<=	Menor o igual	$a <= $b	$a es menor o igual que $b
>=	Mayor o igual	$a >= $b	$a es mayor o igual a $b

Estos operadores se utilizan en combinación con la cláusula WHERE para filtrar los registros según las condiciones especificadas.

Lista de operadores de comparación complejos.

Operador	Nombre	Ejemplo	Devuelve cierto cuando:
LIKE	Parecido	Select * From santos WHERE nombre LIKE 'San%'	Realiza la comparación en función de un patrón, en lugar de un valor concreto.
BETWEEN	Entre	Select * From vacaciones WHERE dias BETWEEN '5'	Encuentra los resultados comprendidos en un rango de números o fechas.
IN	Dentro de	AND '7' Select * From empleados WHERE cargo IN	Parecido al operando =, pero capaz de realizar más de una comparación.

Lista de patrones.

Operador	Nombre	Ejemplo	Resultado
%	Comodín de cadena	S%	*Palabra o frase que empiece por S, seguido de más caracteres.*
_	Comodín de carácter	M_S	Palabra de tres letras que empiece por M y termine por S.

Existen otros operadores de comparación más complejos que realizan un cálculo sobre una cadena de caracteres o una secuencia de números para devolver un resultado lógico de verdadero o falso.

Podemos utilizar, además, patrones con la cláusula LIKE para realizar comparaciones con porciones de textos.

3.5.2 Operadores lógicos

Los operadores lógicos se utilizan para combinar múltiples condiciones en una consulta y evaluar si se cumplen ciertas condiciones lógicas. Su resultado siempre será booleano.

Se utilizan siempre para realizar una operación lógica entre dos operandos de comparación: y su resultado será boleano.

- Hay tres tipos de operadores lógicos: AND, OR y NOT.
- **AND (y).** Devuelve verdadero si todas las condiciones son verdaderas.
- **OR (o)**. Devuelve verdadero si al menos una de las condiciones es verdadera.
- **NOT**. Invierte el resultado de una condición, devolviendo verdadero si la condición es falsa y viceversa.

PARA SABER MÁS: Los valores booleanos son aquellos que corresponden a una numeración binaria. Sólo pueden tomar como valor el 1 o el 0, y suelen asociarse a los valores true (verdadero) o false (falso) respectivamente.

3.5.3 Operadores de concatenación y aritméticos

Los **operadores** de concatenación y aritméticos se utilizan para realizar operaciones de concatenación de cadenas de texto y operaciones aritméticas en una consulta. Algunos de los operadores comunes son:

- **Concatenación (+):** Se utiliza para concatenar dos o más cadenas de texto.
- **Suma (+), Resta (-), Multiplicación (*), División (/):** Se utilizan para realizar operaciones aritméticas en valores numéricos.

Estos operadores se pueden utilizar en la selección de columnas o en las cláusulas WHERE para realizar cálculos o manipulaciones de datos en una consulta.

3.5.4 La precedencia

La precedencia es única para cada operando, y define el orden en el que la consulta calcula las operaciones aritmético-lógicas. Por ejemplo, en la sentencia:

```
SELECT id_Producto, Producto, PVP, Unidades, Descuento, (Unidades * PVP-Descuento) AS Total FROM Stock
```

Primero se realizará la operación PVP*Unidades y del resultado se restará el descuento; eso es así porque la multiplicación y la división tienen preferencia sobre la suma y la resta. Para alterar la precedencia de los operandos, podemos utilizar los paréntesis:

```
SELECT id_Producto, Producto, PVP, Unidades, Descuento, (Unidades * (PVP-Descuento)) AS Total FROM Stock
```

P_Id	Apelli-	Nombre	Número
1	Alonso	Enrique	526
2	Gutiérrez	Pablo	Null
3	Martínez	Eduardo	123

En este caso, primero se restará el descuento al PVP y el resultado se multiplicará a las unidades.

3.6 Tratamiento de valores nulos

Además de las operaciones lógicas, podemos realizar funciones utilizando el **valor nulo** (en inglés NULL), que puede sernos de gran utilidad. El tratamiento de valores nulos es un aspecto importante en consultas, ya que los valores nulos pueden afectar el resultado de una consulta y requieren un manejo especial.

Para empezar, tenemos que saber que sólo algunas de las cláusulas en SQL tienen en cuenta el valor NULL. A continuación, citaremos algunas de las funciones que lo utilizan:

- **IS NULL/IS NOT NULL**. En una consulta, los valores nulos se pueden identificar utilizando la cláusula WHERE y el operador IS NULL o IS NOT NULL.
- **NULLIF (A, B)**. Función que compara dos valores A y B. Si son iguales devuelve NULL, sino devuelve el primer valor; hay que tener siempre en cuenta que el valor A no puede ser nunca NULL.
- **NULLS FIRST**. Por defecto, cuando usamos el ORDER BY, el resultado coloca los valores NULL al fi nal. En cambio, con la sentencia ORDER BY NULLS FIRST aparecerán los NULLS en los primeros lugares de los registros consultados.
- **COALESCE (A,B,C, [...])**. La función devuelve el primer valor de la lista que no sea NULL.

Veamos a continuación cómo obtener el resultado de una consulta ordenada por el parámetro Número, pero mostrando primero los resultados nulos:

```
SELECT Apellido, Nombre, Numero
FROM NombreDeTabla
ORDER BY Apellido NULLS FIRST;
```

Y como resultado obtendremos la tabla:

P_Id	Apelli-	Nombre	Número
2	Gutiérrez	Pablo	Null
3	Martínez	Eduardo	123
1	Alonso	Enrique	526

3.7 Consultas de resumen y funciones de agregado

3.7.1 Utilización de funciones de agregado (SUM, COUNT, AVG, etc.)

Las funciones de agregado son herramientas poderosas que nos permiten realizar cálculos sobre conjuntos de datos y obtener resultados resumidos. Algunas de las funciones de agregado más comunes son:

- **SUM:** Calcula la suma de los valores en una columna numérica.
- **COUNT:** Cuenta el número de filas en un conjunto de datos.
- **AVG**: Calcula el promedio de los valores en una columna numérica.
- **MAX:** Devuelve el valor máximo de una columna.
- **MIN:** Devuelve el valor mínimo de una columna.

Estas funciones se utilizan en combinación con la cláusula SELECT para realizar cálculos sobre conjuntos de datos. Por ejemplo, para calcular la suma de los valores de una columna "precio" en una tabla "productos", podemos utilizar la siguiente consulta:

```
SELECT SUM(precio) FROM productos;
```

Esto devolverá el resultado de la suma de los valores de la columna "precio".

3.7.2 Generación de consultas de resumen mediante GROUP BY

La cláusula GROUP BY nos permite agrupar filas de datos basándonos en los valores de una o más columnas. Esto es útil cuando queremos realizar cálculos de resumen para grupos específicos de datos.

Al utilizar la cláusula GROUP BY, podemos combinarla con funciones de agregado para obtener resultados resumidos por grupo. Por ejemplo, supongamos que tenemos una tabla "ventas" con las columnas "producto", "cantidad" y "precio", y queremos calcular la cantidad total de cada producto vendido. Podemos utilizar la siguiente consulta:

```
SELECT producto, SUM(cantidad) FROM ventas GROUP BY producto;
```

Esto nos devolverá los resultados agrupados por producto, mostrando el nombre del producto y la suma de la cantidad vendida para cada uno.

La cláusula GROUP BY nos permite obtener información resumida y analítica sobre nuestros datos, dividiéndolos en grupos y aplicando funciones de agregado a cada grupo. Esto resulta útil en situaciones en las que necesitamos realizar análisis de datos por categorías o segmentos específicos.

Un ejemplo del funcionamiento lo podemos encontrar en la siguiente imagen donde encontramos una consulta que nos devolverá el equipo y la media de edad de los jugadores de ese equipo, pero solo para aquellos equipos que tengan más de un jugador cuya edad sea mayor que 24.

Equipo	Edad	Nombre
Bcn	26	Álex
Vlc	26	Pablo
Mad	22	Javi
Vlc	23	Sergio
Vlc	25	Raúl
Mad	26	Carlos
Bcn	27	Manu
Bcn	28	Marc

```
SELECT Equipo, AVG(Edad)
FROM jugadores
WHERE Edad>24
GROUP BY Equipo
HAVING COUNT(nombre)>1
```

Equipo	Edad	Nombre
Bcn	26	Álex
Vlc	26	Pablo
Vlc	25	Raúl
Bcn	27	Manu
Bcn	28	Marc

PARA SABER MÁS: La cláusula GROUP BY realiza la agrupación de registros en función de unas características comunes. Posteriormente la cláusula HAVING aplica una consulta de selección de registros agrupados anteriormente.

3.8 Agrupamiento de registros

El agrupamiento de registros es una funcionalidad importante en las consultas de bases de datos que nos permite agrupar filas en función de los valores de una o varias columnas específicas.

Esto nos permite realizar cálculos y obtener resúmenes de datos basados en grupos específicos. A continuación, veamos cómo funcionan las cláusulas implicadas en este tipo de consultas.

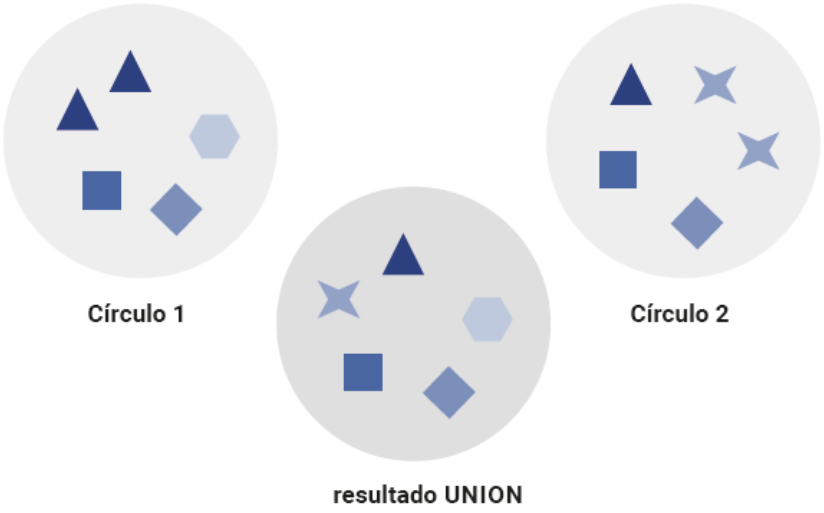
GROUP BY (agrupar por). Con esta cláusula agruparemos el conjunto de filas que tengan el mismo valor en la columna que hayamos definido. Se sitúa después de la cláusula WHERE y antes de ORDER BY, y se aplica a los resultados obtenidos después de aplicar la condición WHERE.

HAVING (teniendo). La cláusula HAVING se utiliza junto con la cláusula GROUP BY en consultas agrupadas para filtrar los resultados basados en condiciones que involucran funciones de agregado. Mientras que la cláusula WHERE se utiliza para filtrar filas individuales antes de la agrupación, la cláusula HAVING se aplica después de la agrupación y permite filtrar los grupos resultantes.

La sintaxis básica de la cláusula HAVING es la siguiente:

```
SELECT columna1, columna2, funcion_agregado(columna3)
FROM tabla
WHERE condiciones
GROUP BY columna1, columna2
HAVING condiciones_agregado;
```

A continuación, mostramos una representación de una base da datos que almacena un listado de jugadores de fútbol, su edad y el equipo al que pertenecen. En ella puede verse cómo debemos realizar una consulta para obtener el nombre de cada equipo y la edad media de los jugadores mayores de 24 años. Además, sólo se muestran los equipos que contengan más de un jugador mayor de 24 años



Cuando realizamos agrupamiento de registros, es común utilizar funciones de agregado para realizar cálculos en los grupos. Algunas funciones de agregado comunes son SUM, COUNT, AVG, MAX, MIN, entre otras. Estas funciones se aplican a una columna específica y proporcionan un valor resumido para cada grupo. Por ejemplo:

```
SELECT columna1, funcion_agregado(columna2)
FROM tabla
WHERE condiciones
GROUP BY columna1;
```

En este ejemplo, aplicamos la función de agregado a la columna "columna2" y obtenemos el resultado resumido por cada grupo definido por la columna "columna1".

3.9 Unión de consultas

Podemos obtener el resultado de dos consultas diferentes y mostrarlas al mismo tiempo al utilizar las llamadas consultas de unión. Para ello emplearemos la cláusula UNION, que se utiliza para combinar los resultados de múltiples consultas en una sola tabla de resultados. Cada consulta dentro de la cláusula **UNION** debe tener la misma estructura de columnas y el mismo tipo de datos. El resultado de la consulta UNION incluirá todas las filas distintas de las consultas combinadas, como si utilizáramos SELECT DISTINCT.

A continuación, veamos una representación configuras geométricas del contenido de dos tablas, y cómo el resultado de la consulta UNION devuelve todas las figuras, sin repetirlas, que existen tanto en la "TABLA 1" como en la "TABLA 2"

La consulta que representa este grafico sería:

```
SELECT * FROM TABLA 1

UNION

SELECT * FROM TABLA 2
```

En esta consulta, se seleccionarán todas las filas y columnas de TABLA1 y se combinarán con todas las filas y columnas de TABLA2. El resultado será una única tabla que incluirá todas las filas distintas de ambas tablas.

Es importante tener en cuenta que las tablas involucradas en la consulta deben tener la misma estructura de columnas y el mismo tipo de datos para que la combinación sea exitosa. Si las tablas difieren en cuanto a la estructura de columnas, es posible que debas especificar manualmente las columnas que deseas seleccionar en lugar de utilizar el asterisco (*).

Además, ten en cuenta que la cláusula UNION eliminará las filas duplicadas en el resultado final. Si deseas incluir todas las filas, incluidas las filas duplicadas, debes utilizar la cláusula UNION ALL en lugar de UNION.

```
SELECT *

FROM Equipos

INNER JOIN jugadores

ON equipo.id_equipo=jugadores.id_equipos
```

3.10 Composiciones internas

Las composiciones internas, también conocidas como inner joins, son utilizadas para combinar filas de dos o más tablas en una consulta en función de una condición de igualdad entre las columnas relacionadas. Este tipo de composiciones se utilizan cuando se desea obtener únicamente las filas que coinciden en ambas tablas.

```
SELECT *

FROM Equipos

INNER JOIN jugadores

ON equipo.id_equipo=jugadores.id_equipos
```

La sintaxis básica para realizar una composición interna es la siguiente:

```
SELECT columnas
FROM tabla1
INNER JOIN tabla2
ON tabla1.columna = tabla2.columna;
```

En este ejemplo, tabla1 y tabla2 son los nombres de las tablas que se desean combinar, y columna es la columna en la cual se basará la igualdad para unir las filas. El resultado de la consulta será un conjunto de filas que cumplen con la condición de igualdad especificada.

Veamos en el siguiente ejemplo la utilización de un INNER JOIN: en una tabla con las siguientes personas, vemos que la columna DNI tiene un valor único para cada registro, puesto que es la clave primaria que identifica cada una de las entradas.

Tabla en la que la columna DNI es la clave primaria

DNI	Apelli-	Nombre	Dirección	Ciudad
00000001A	Sánchez	Pablo	Gran Vía	Madrid
00000002B	Martínez	Eduardo	Diagonal	Barcelona
00000003B	Pérez	Enrique	Paseo Marítimo	Málaga

Tabla Registro en la que la columna DNI no está como clave primaria

R_Id	RegistroNo	DNI
1	77895	00000001A
2	44678	00000001A
3	22456	00000002B
4	24562	00000002B
5	34764	00000005D

Tenemos también una tabla llamada Registro:

Vemos que DNI también está en esta tabla, pero no como clave primaria. Aun así, la utilizaremos para relacionarla con la tabla Personas:

```
SELECT Personas.Apellido, Personas.Nombre, Registro.RegistroNo
FROM Personas
INNER JOIN Registro
ON Personas.DNI= Registro.DNI
ORDER BY Personas.Apellido
```

El resultado de la consulta se muestra en la siguiente tabla:

Resultado de la consulta con la cláusula INNER JOIN

Apellido	Nombre	OrderNo
Sánchez	Pablo	77895
Sánchez	Pablo	44678
Martínez	Eduardo	22456
Martínez	Eduardo	24562

El INNER JOIN devuelve los registros que tienen al menos una coincidencia entre ambas tablas en función del DNI, que es el parámetro especifi cado en la cláusula "ON". Si hay algún registro en Personas que no se encuentra en Órdenes, esta fila no se mostrará.

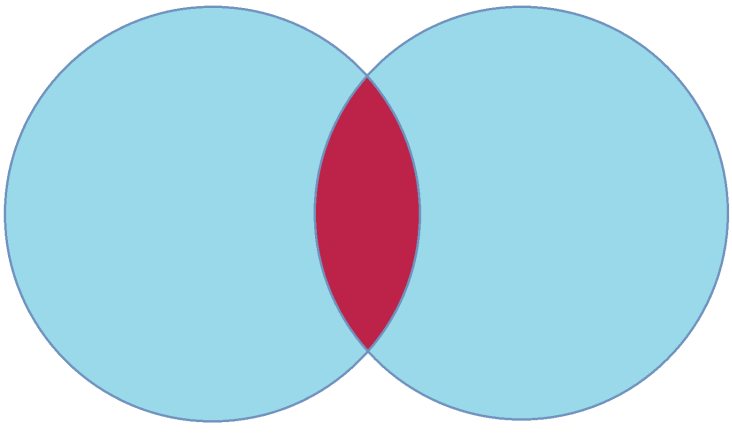
La operación que acabamos de realizar también podría conseguirse con la siguiente consulta sin utilizar INNER JOIN:

```
SELECT Personas.Apellido, Personas.Nombre, Registro.RegistroNo

FROM Personas, Registro

WHERE Personas.DNI= Registro.DNI

ORDER BY Personas.Apellido
```

Es importante remarcar que, aunque una consulta INNER JOIN también puede realizarse con una consulta WHERE, esta última tiene un rendimiento mucho menor, y en tablas de gran tamaño puede incrementar el tiempo de ejecución de manera considerable.

Esto es debido a que con la cláusula WHERE se realiza la consulta de todos los registros de ambas tablas y se realiza el producto cartesiano (cada registro de A lo relacionamos con cada registro de B), y después ese resultado lo filtramos, mientras que con el INNER JOIN primero se fi ltran los resultados de cada tabla y después simplemente los une, sin tener que realizar el producto cartesiano.

La clase NATURAL JOIN es una abreviación de INNER JOIN por condición de igualdad. Además, elimina las columnas que estén repetidas.



Representación de dos círculos. La parte en rojo representa los datos comunes de ambos círculos.

3.10.1. Nombres cualificados

Si realizamos una consulta en una sola tabla, el sistema deduce que la columna a la que queremos acceder es la tabla que hemos definido con la cláusula FROM. Cuando trabajamos con diferentes tablas o bases de datos, es necesario utilizar nombres cualificados: "nombre_tabla.nombre_columna".

Se utiliza el punto para separar el nombre del objeto y el nombre de su contenedor, con el fin de crear una cadena de elementos que representa el objeto y el lugar al que pertenece.

Tablas de ejemplo utilizadas en los ejercicios de OUTER JOIN

Tabla Empleado		Tabla Departamento	
Apellido	**IDDepartamento**	**Nombre Departamento**	**IDDepartamento**
Sánchez	31	Dirección	31
Martínez	33	Informática	33
Gutiérrez	33	Comercial	34
Pérez	34	Contable	35
Moreno	34		
García	36		

3.11 Composiciones externas

```
SELECT *
FROM        empleado
            LEFT OUTER JOIN departamento
                ON empleado.IDDepartamento = departamento.IDDepartamento
```

Resultado de la ejecución de la cláusula LEFT

Empleado. Apellido	Empleado. IDDepartamento	Departamento. Nombre Departamento	Empleado. IDDepartamento
Martínez	33	Informática	33
Sánchez	31	Dirección	31
Pérez	34	Comercial	34
Moreno	34	Comercial	34
García	36	NULL	NULL
Gutiérrez	33	Informática	33

Las composiciones externas se utilizan para realizar consultas uniendo dos o más tablas; pero, a diferencia de las composiciones internas (INNER JOIN), no es necesario que cada registro de una tabla tenga una correspondencia con las demás. El registro se mantiene en la tabla combinada, rellenando las columnas que no tengan información con valores

NULL.

```
SELECT *
FROM        empleado
            RIGHT OUTER JOIN departamento
                ON empleado.IDDepartamento = departamento.IDDepartamento
```

Resultado de la ejecución de la cláusula RIGHT

Empleado. Apellido	Empleado. IDDepartamento	Departamento. Nombre Departamento	Empleado. IDDepartamento
Pérez	34	Comercial	34
Martínez	33	Informática	33
Moreno	34	Comercial	34
Gutiérrez	33	Informática	33
Sánchez	31	Dirección	31
NULL	NULL	Contable	35

Para crear la composición externa, usaremos la cláusula OUTER JOIN (unión externa) junto a los parámetros RIGHT (derecha), LEFT (izquierda) o FULL (todo).

Veamos a continuación las tablas, que utilizaremos para realizar los ejemplos de los diferentes tipos de OUTER JOIN:

Tabla Empleado		Tabla Departamento	
Apellido	**IDDepartamento**	**Nombre Departamento**	**IDDepartamento**
Sánchez	31	Dirección	31
Martínez	33	Informática	33
Gutiérrez	33	Comercial	34
Pérez	34	Contable	35
Moreno	34		
García	36		

```
SELECT *
FROM        empleado
            FULL OUTER JOIN departamento
                ON empleado.IDDepartamento = departamento.IDDepartamento
```

Resultado de la ejecución de la cláusula FULL

Empleado. Apellido	Empleado. IDDepartamento	Departamento. Nombre Departamento	Empleado. IDDepartamento
Pérez	34	Comercial	34
Martínez	33	Informática	33
Moreno	34	Comercial	34
García	36	NULL	NULL
Gutiérrez	33	Informática	33
Sánchez	31	Dirección	31
NULL	NULL	Contable	35

- **LEFT**. Realiza una composición externa izquierda, es decir, incluye todas las filas de la tabla izquierda y las filas de la tabla derecha que coinciden en la condición de igualdad. Si no hay coincidencias, se incluirán NULL en las columnas correspondientes de la tabla derecha.

Empleado. Apellido	Empleado. IDDepartamento	Departamento. Nombre Departamento	Empleado. IDDepartamento
Martínez	33	Informática	33
Sánchez	31	Dirección	31
Pérez	34	Comercial	34
Moreno	34	Comercial	34
García	36	NULL	NULL
Gutiérrez	33	Informática	33

Podemos ver como el registro García de la tabla Empleado aparece en el resultado relacionado con valores NULL al no haber una correspondencia con la tabla Departamento.

- **RIGHT**. Realiza una composición externa derecha, incluyendo todas las filas de la tabla derecha y las filas de la tabla izquierda que coinciden en la condición de igualdad. Si no hay coincidencias, se incluirán NULL en las columnas correspondientes de la tabla izquierda.

Empleado. Apellido	Empleado. IDDepartamento	Departamento. Nombre Departamento	Empleado. IDDepartamento
Pérez	34	Comercial	34
Martínez	33	Informática	33
Moreno	34	Comercial	34
Gutiérrez	33	Informática	33
Sánchez	31	Dirección	31
NULL	NULL	Contable	35

Podemos ver como el registro Contable de la tabla Departamento aparece en el resultado, relacionado con valores NULL, al no haber una correspondencia con la tabla Empleado.

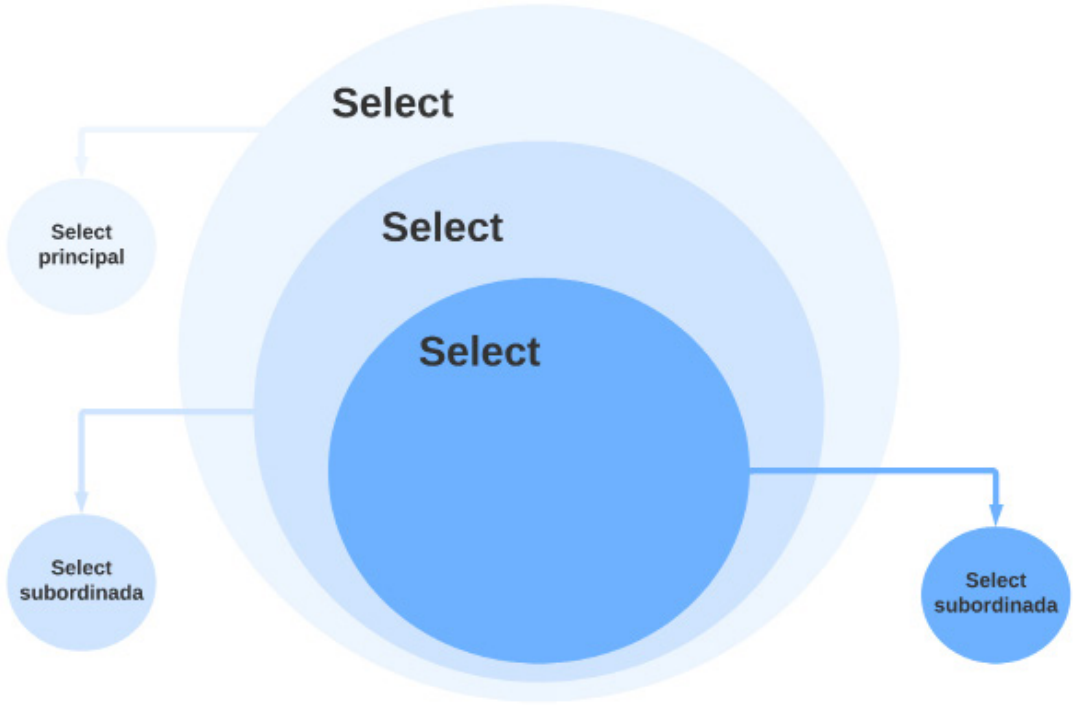
- **FULL**. Realiza una composición externa completa, incluyendo todas las filas de ambas tablas, independientemente de si hay coincidencias o no. Si no hay coincidencias, se incluirán NULL en las columnas correspondientes de la tabla contraria.

Empleado. Apellido	Empleado. IDDepartamento	Departamento. Nombre Departamento	Empleado. IDDepartamento
Pérez	34	Comercial	34
Martínez	33	Informática	33
Moreno	34	Comercial	34
García	36	NULL	NULL
Gutiérrez	33	Informática	33
Sánchez	31	Dirección	31
NULL	NULL	Contable	35

En este caso, el registro García aparece en el resultado relacionado con valores NULL, así como el departamento Contable, que también tiene valores nulos. Si hubiéramos realizado esta misma consulta con un inner join, los resultados con valores null no hubieran aparecido en el resultado.

3.12 Subconsultas. Ubicación de subconsultas. Subconsultas anidadas

Las subconsultas son consultas que se incluyen dentro de una consulta principal y se utilizan para obtener información adicional o realizar cálculos específicos. Se ubican dentro de cláusulas como SELECT, FROM, WHERE, HAVING o JOIN, y su resultado se utiliza en la consulta principal, formando de esta manera una cadena de consultas anidadas.



Representación gráfica de las consultas anidadas.

Las subconsultas se pueden dividir en tres grupos:

Subconsultas escalonadas:

Las subconsultas escalonadas se utilizan para obtener un valor que se utilizará como filtro en la cláusula WHERE y/o HAVING de la consulta principal. Estas subconsultas devuelven un único valor y se pueden utilizar operadores de comparación tradicionales como menor que (<), menor o igual que (<=), mayor que (>), mayor o igual que (>=), distinto de (<>), o igual a (=).

Un ejemplo de una subconsulta escalonada sería:

```
SELECT nombre
FROM empleados
WHERE salario > (SELECT AVG(salario) FROM empleados);
```

En este ejemplo, la subconsulta escalonada "(SELECT AVG(salario) FROM empleados)" calcula el salario promedio de todos los empleados. Luego, la consulta principal selecciona los nombres de los empleados cuyo salario es mayor que el salario promedio calculado en la subconsulta.

Subconsulta de lista

Las subconsultas de lista se utilizan cuando la subconsulta devuelve más de un valor o cuando se espera que pueda devolver varios valores (si no se ha probado previamente). En este tipo de subconsultas, se utilizan operadores de búsqueda literales (ALL, ANY, EXISTS) o de comparación literales (IN) para determinar qué hacer con el conjunto de datos devuelto por la subconsulta.

Aquí tienes un ejemplo de una subconsulta de lista utilizando el operador IN:

```
SELECT nombre

FROM empleados

WHERE departamento_id IN (SELECT departamento_id FROM departamentos WHERE ubicacion = 'Madrid');
```

En este ejemplo, la subconsulta "(SELECT departamento_id FROM departamentos WHERE ubicacion = 'Madrid')" devuelve un conjunto de valores de los ID de departamento que se encuentran en la ubicación 'Madrid'. Luego, la consulta principal selecciona los nombres de los empleados cuyos departamentos están en esa lista de ID de departamento.

Operador ALL

Se utiliza para evaluar si todos los valores de la subconsulta cumplen una condición específica. Por ejemplo:

```
SELECT nombre

FROM empleados

WHERE salario > ALL (SELECT salario FROM empleados WHERE departamento = 'Ventas');
```

En este ejemplo, se seleccionan los nombres de los empleados cuyo salario es mayor que todos los salarios obtenidos en la subconsulta.

Operador ANY

Se utiliza para evaluar si alguno de los valores de la subconsulta cumple una condición específica. Por ejemplo:

```
SELECT nombre

FROM empleados

WHERE salario > ANY (SELECT salario FROM empleados WHERE departamento = 'Ventas');
```

En este caso, se seleccionan los nombres de los empleados cuyo salario es mayor que al menos uno de los salarios obtenidos en la subconsulta.

Operador EXISTS

Se utiliza para verificar si existe al menos una fila que cumpla una condición en otra tabla. Por ejemplo:

```
SELECT nombre

FROM empleados

WHERE EXISTS (SELECT * FROM ventas WHERE ventas.id_empleado = empleados.id);
```

En este ejemplo, se seleccionan los nombres de los empleados que tienen al menos una venta asociada en la tabla de ventas.

Operador IN

Se utiliza para verificar si un valor coincide con alguno de los valores de la subconsulta. Por ejemplo:

```
SELECT nombre

FROM empleados

WHERE departamento IN (SELECT id_departamento FROM departamentos

WHERE ubicacion = 'Madrid');
```

En este caso, se seleccionan los nombres de los empleados cuyo departamento coincide con uno de los departamentos obtenidos en la subconsulta.

4. Resumen

Los SGDB (Sistemas gestores de Bases de datos) proporcionan a usuarios y administradores la posibilidad de manejar las bases de datos de una manera gráfica e intuitiva. Suponen una mejora importante en la que no es necesario el conocimiento experto en programación de bases de datos para poder realizar consultas o tareas de mantenimiento en la base de datos.

Las consultas son leídas por las bases de datos para poder realizar gestiones de mantenimiento, inserción de registro u obtención de la información almacenada en las bases de datos. La consulta SELECT es utilizada por los usuarios para obtener datos almacenados, seleccionados según los criterios y condiciones que acompañan la instrucción.

Dentro de las opciones de las consultas, es posible obtener resultados en un determinado orden o extraer aquella información que entra dentro de los parámetros establecidos por el usuario. Estos parámetros son definidos mediante operadores de comparación, lógicos o de precedencia. Otra de las posibilidades de la sentencia SELECT consiste en anidar consultas, con lo que se consigue que una petición sea aplicada sobre el resultado de una consulta anterior.

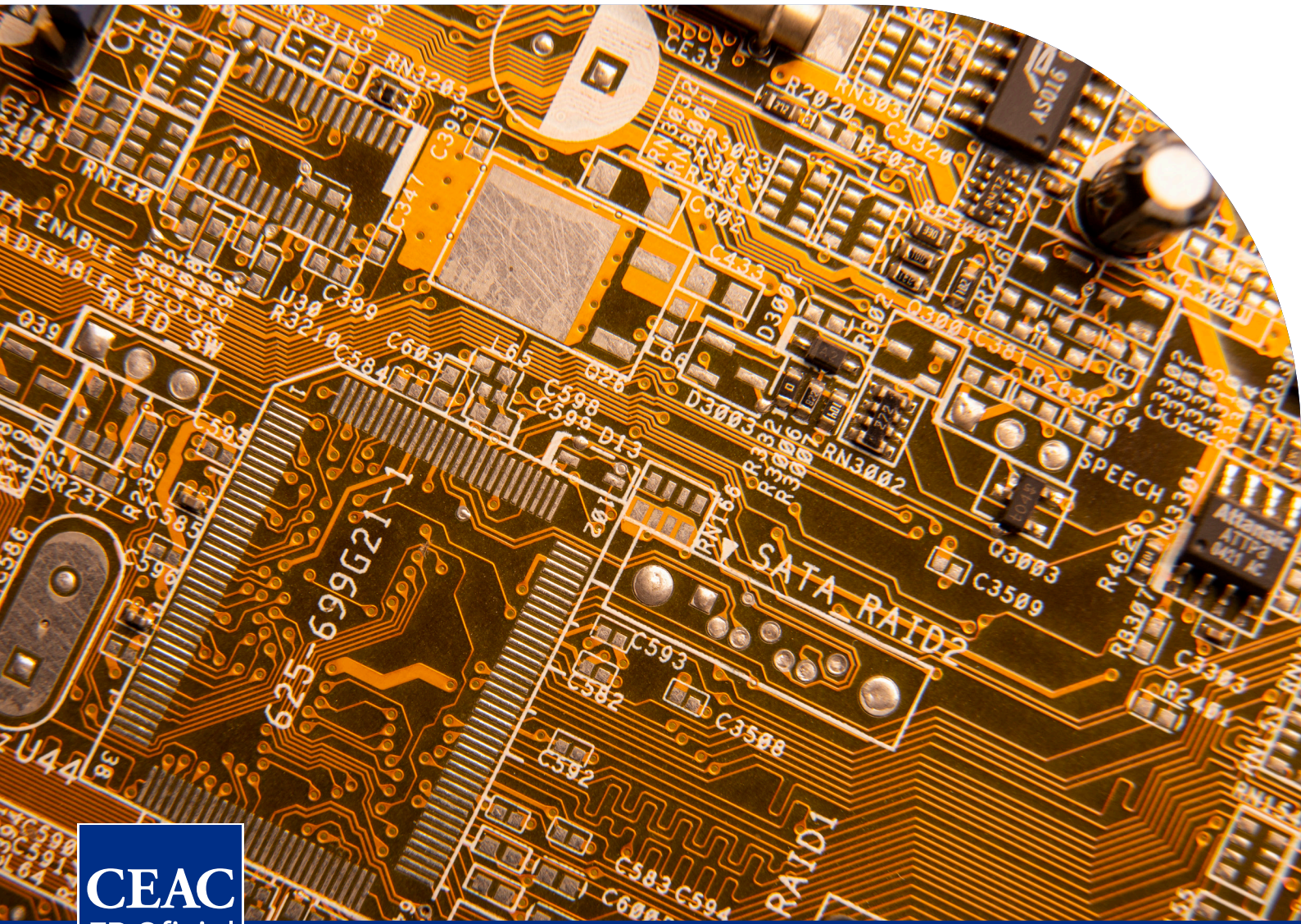
Desarrollo de aplicaciones multiplataforma

MÓDULO:

Base de datos

UNIDAD:

Tratamiento de datos



CEAC
FP Oficial

MÓDULO:

Base de datos

UNIDAD:

Tratamiento de datos

CEAC FP Oficial

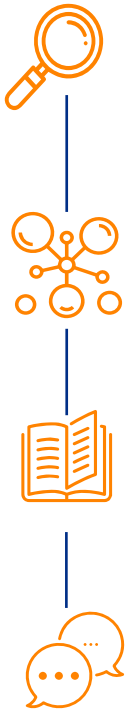
ÍNDICE

1. Introducción

En este apartado analizaremos el tratamiento de datos, que consiste en introducir, modificar y eliminar la información de las bases de datos. Sin embargo, en el tratamiento de datos no sólo influyen factores como la simple obtención de información, sino también los permisos y los problemas de acceso simultáneo.

2. Objetivos pedagógicos

Los objetivos de aprendizaje que se pretenden alcanzar con esta unidad son:



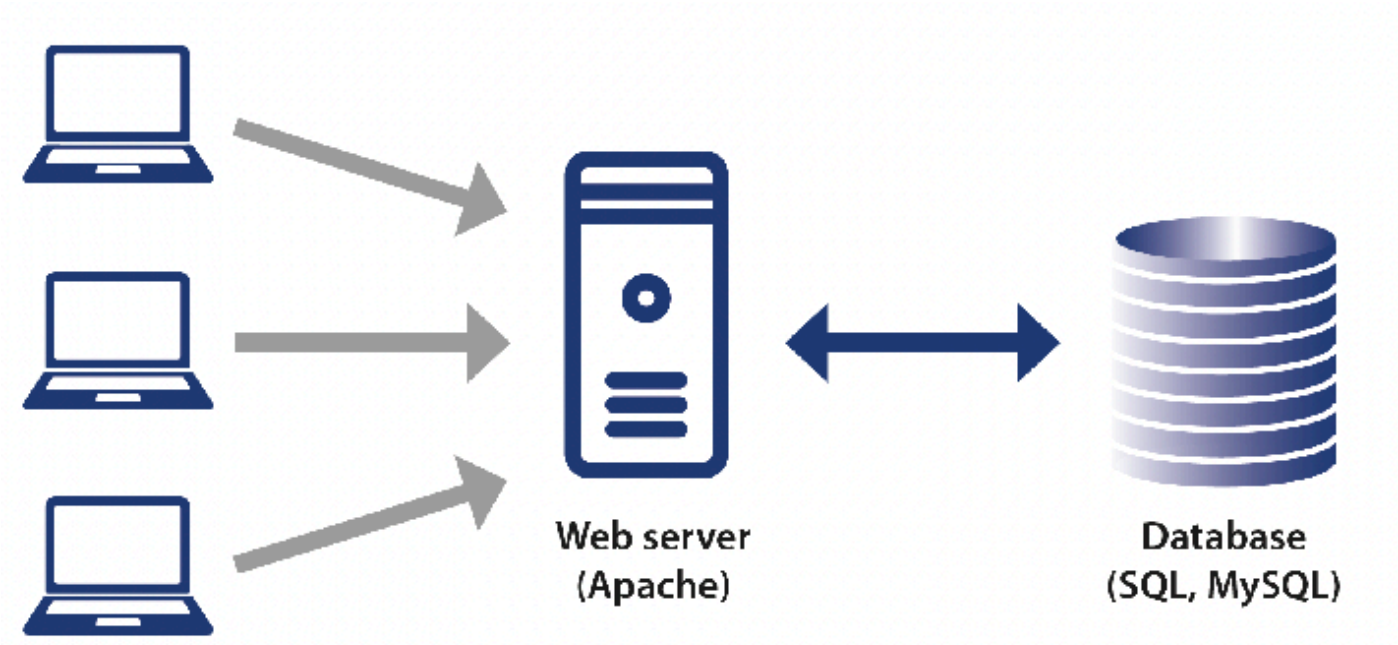
- **Comprender el concepto de tratamiento de datos.**
- **Identificar los componentes clave del tratamiento de datos.**
- **Explorar los desafíos de seguridad y permisos.**
- **Entender el impacto del acceso simultáneo.**

3. Tratamiento de datos

3.1 Tratamiento de datos

Para poder realizar consultas a bases de datos, deberemos previamente haber introducido toda la información necesaria de manera estructurada y ordenada, teniendo en cuenta los numerosos factores relacionados con los permisos de acceso a la información y evitando la modificación simultánea de los datos. Podemos definir el tratamiento de datos en tres fases principales:

- Inserción de registros. Se trata de subir a la base de datos toda la información que queremos almacenar. Deberemos tener en cuenta el tipo de datos que acepta la base de datos (número, cadena de caracteres, etc.) y también el tamaño de éstos, puesto que tendremos un espacio limitado para albergar los datos, como ocurre en el ejemplo de la imagen.
- Modificación de registros. Consiste en manipular la información ya existente en la base de datos con el fin de modificar los datos a consultar.
- Eliminación de registros. Para el buen mantenimiento de la base de datos es conveniente proceder a la eliminación de registros innecesarios con el fin de liberar espacio en el disco duro y mejorar la indexación.



Introducción de información de varios usuarios en la base de datos.

3.2 Herramientas gráficas proporcionadas por el sistema gestor para la edición de la información

El tratamiento de los datos puede resultar complicado debido a una serie de factores que deberemos tener en cuenta, como es el caso de la eliminación de datos dependientes. Por ello, los sistemas gestores proporcionan **herramientas graficas** para poder realizar las funciones de mantenimiento y tratamiento de la información de manera sencilla e intuitiva a fin de evitar al usuario tener que utilizar consultas construidas a mano.

Los datos se representan generalmente en forma de tabla, lo que ofrece al usuario la opción de insertar nuevos registros. En ese caso, aparecerá en nuestra tabla una nueva fila con la información que acabamos de añadir.

Una de las principales ventajas que ofrecen las herramientas gráficas es el tratamiento de la información por bloques; es decir, podemos crear, eliminar y modificar varios registros en un solo paso, en lugar de realizar varias consultas para cada una de las filas.

PARA SABER MÁS: Una vez insertado un nuevo registro, solo podremos introducir nueva información mediante una consulta de modificación, puesto que la sentencia INSERT sirve para crear nuevos registros.

3.3 Inserción de registros. Inserciones a partir de una consulta

A pesar de disponer de las herramientas graficas para el tratamiento de la información, es preciso saber cómo podremos modificar los datos mediante la utilización de las consultas. Deberemos utilizar dichas consultas cuando tengamos que insertar nueva información mientras ejecutamos un programa. La consulta encargada de insertar información en una base de datos se llama **INSERT** **(insertar)**.

Veamos a continuación cuál es la sintaxis básica de una consulta INSERT:

- **INSERT INTO (insertar en).** Crea un nuevo registro. Seguido de la sentencia INSERT INTO definiremos la tabla en la que se creará el registro, y enumeraremos entre paréntesis los campos en los que queremos insertar los datos.

  INSERT INTO [nombre tabla](columna1,columna2)

  VALUES (valor1,valor2)

- **VALUES (valores).** Introduciremos entre paréntesis los valores con los que queramos inicializar el registro que estamos creando.

  Cuando ejecutamos la consulta INSERT, crearemos una nueva fila en la tabla definida. Deberemos introducir los datos de todas las columnas que componen dicha tabla, pues, de lo contrario, los datos que insertemos obtendrían el valor NULL (nulo), que, a veces, puede producir un error, ya que no todos los campos aceptan este valor. Veamos en la siguiente tabla un ejemplo de cómo insertar un registro.

Tabla PERSONAS antes de insertar la fila

P_Id	Apellido	Nombre	Ciudad
1	Sánchez	Pablo	Barcelona
2	Martínez	Sergio	Madrid
3	Pérez	Álex	Santander

Para insertar una nueva persona en nuestra tabla bastará conv ejecutar la siguiente sentencia:

```
INSERT INTO PERSONAS
P_Id, Apellido, Nombre, Ciudad
values
4,Pérez, Juan, Málaga
```

Tabla PERSONAS después de insertar la fila

P_Id	Apellido	Nombre	Ciudad
1	Sánchez	Pablo	Barcelona
2	Martínez	Sergio	Madrid
3	Pérez	Álex	Santander
4	Pérez	Juan	Málaga

3.4 Borrado y modificación de registros

Otros tratamientos de datos fundamentales para el mantenimiento de la base de datos es el **borrado de registros** innecesarios y la **modificación de registros** para actualizar la información almacenada.

3.4.1 Borrado de registros

El **borrado de registros** es una operación que muchas veces se pasa por alto, pues se tiende a creer que si una determinada información no nos interesa basta con no acceder a ella. Sin embargo, eso comporta una masificación de la base de datos, que, con el tiempo, puede almacenar gran cantidad de información innecesaria y ocupar un considerable espacio de almacenamiento.

Por ello, una vez nos hayamos asegurado de que los datos son realmente innecesarios es conveniente proceder a su eliminación. Para ello, podemos realizar el borrado desde las herramientas gráficas proporcionadas por el sistema gestor o mediante la consulta **DELETE (eliminar)**:

```
DELETE FROM [tabla]
WHERE [condición]
```

- **DELETE FROM.** Elimina un registro que se encuentra en la tabla definida.
- **WHERE**. Definiremos las condiciones que tienen que cumplir los registros que deseamos borrar. Es especialmente útil en los casos en los que los registros tienen un número de identificación único, para que la consulta aplique las modificaciones de los datos sólo en el registro que deseamos. Podremos eliminar más de un registro en una misma consulta si sus datos coinciden con la condición establecida.

Tabla PERSONAS antes de borrar la fila

P_Id	Apellido	Nombre	Ciudad
1	Sánchez	Pablo	Barcelona
2	Martínez	Sergio	Madrid
3	Pérez	Álex	Santander
4	Pérez	Juan	Málaga

En la misma tabla obtenida en el apartado anterior, vamos a ejecutar una sentencia para eliminar uno de sus registros. Según esta sentencia:

```
DELETE FROM Personas
WHERE Apellido='Pérez' AND Nombre='Juan'
```

Obtendremos la siguiente tabla:

Tabla PERSONAS después de borrar la fila

P_Id	Apellido	Nombre	Ciudad
1	Sánchez	Pablo	Barcelona
2	Martínez	Sergio	Madrid
3	Pérez	Álex	Santander

Si a la tabla original le aplicamos esta sentencia:

```
DELETE FROM Personas
WHERE Apellido='Pérez' OR Apellido='Martínez'
```

Obtendremos la siguiente tabla:

Tabla PERSONAS después de borrar varias filas

P_Id	Apellido	Nombre	Ciudad
1	Sánchez	Pablo	Barcelona

3.4.2 Modifi cación de registros

Una vez creados los registros resultará indispensable disponer de la opción de modificar dicha información. Desde las herramientas gráficas, podremos modificar el contenido de los registros, pero también podremos utilizar la consulta UPDATE (actualizar):

- **UPDATE**. Modifica los datos de un registro. Seguida de esta cláusula definiremos la tabla en la que se encuentran los registros que deseamos modificar.

```
UPDATE [nombre tabla]
SET [campo1] = [valor1],[campo2] = [valor2],
WHERE [condición del registro]
```

- **SET**. Definiremos tanto el campo que queremos modificar como el nuevo valor que deseamos introducir. Podremos definir más de un campo que deseemos modifi car, separándolo por comas.
- **WHERE**. Definiremos las condiciones que tienen que cumplir los registros que deseamos modificar. Es especialmente útil en los casos en los que los registros tienen un número de identificación único, con el fin de que las consultas apliquen las modificaciones de los datos sólo al registro que deseemos. Podremos modificar más de un registro en una misma consulta si sus datos coinciden con la condición establecida.

  A modo de ejemplo, tenemos las siguientes tablas:

Tabla de ciudades y países

Ciudad	País
Barcelona	España
Madrid	Alemania
Santander	Francia
Málaga	Rusia

Tabla de personas

P_Id	Apellido	Nombre	Ciudad	País
1	Sánchez	Pablo	Barcelona	España
2	Martínez	Sergio	Madrid	Alemania
3	Pérez	Álex	Santander	Francia
4	Pérez	Juan	Málaga	España

Podemos aplicar la sentencia UPDATE haciendo referencia a las dos tablas:

```
UPDATE Personas, Ciudades
SET Pais='España'
WHERE Ciudad='Madrid'
```

Veremos entonces que en las dos tablas ha cambiado el País por España en los registros en los que el valor de la ciudad es Madrid.

Tabla de ciudades y países antes de actualizar las filas

Ciudad	País
Barcelona	España
Madrid	España
Santander	Francia
Málaga	España

Tabla de personas antes de actualizar las filas

P_Id	Apellido	Nombre	Ciudad	País
1	Sánchez	Pablo	Barcelona	España
2	Martínez	Sergio	Madrid	España
3	Pérez	Álex	Santander	Francia
4	Pérez	Juan	Málaga	España

Si no ponemos ninguna condición al ejecutar la sentencia UPDATE, se cambiarán todos los registros de la tabla:

```
UPDATE Personas, Ciudades

SET Pais='España'
```

Tabla de ciudades y países después de actualizar las filas

Ciudad	País
Barcelona	España
Madrid	España
Santander	España
Málaga	España

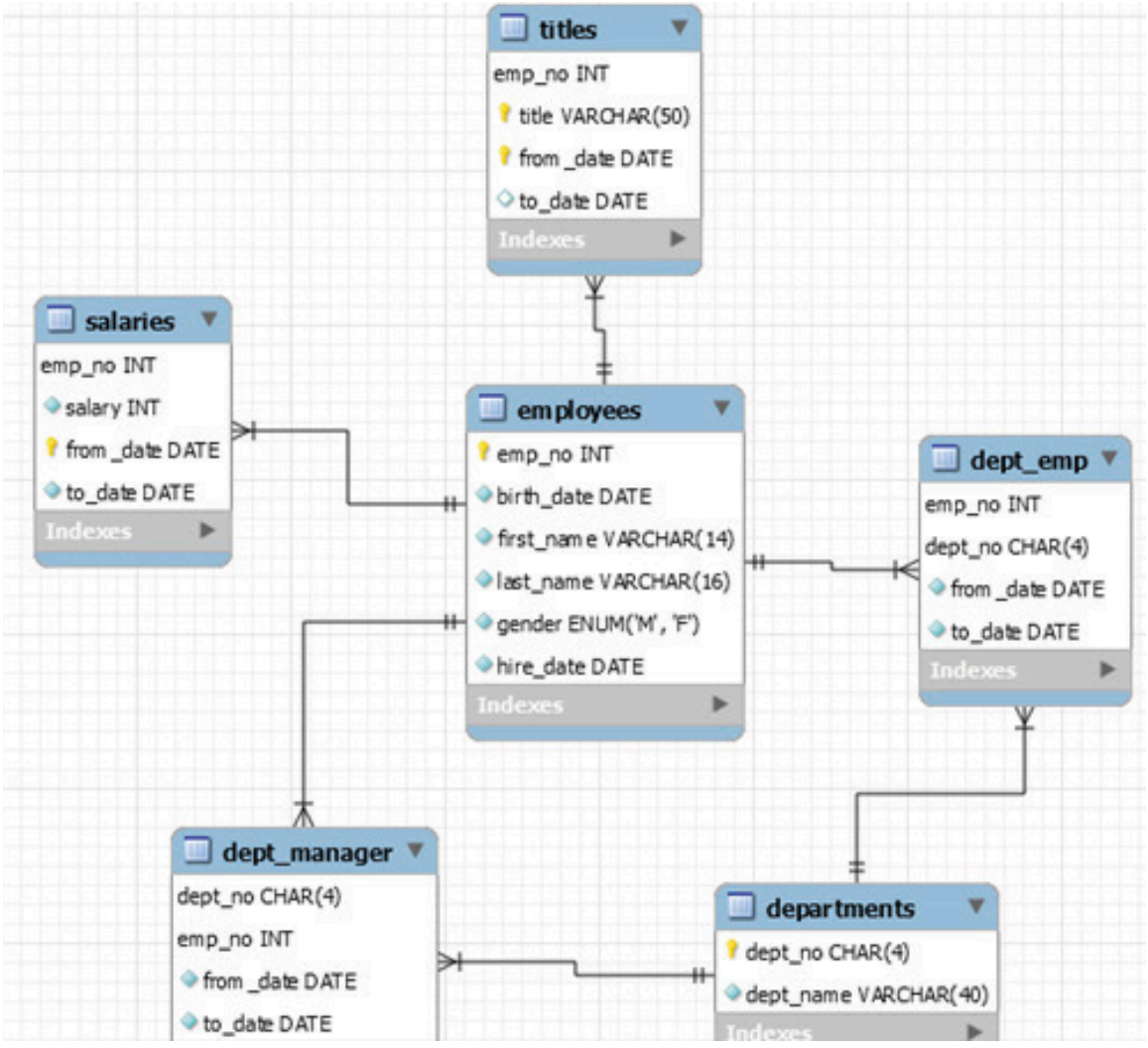
Tabla de personas después de actualizar las filas

P_Id	Apellido	Nombre	Ciudad	País
1	Sánchez	Pablo	Barcelona	España
2	Martínez	Sergio	Madrid	España
3	Pérez	Álex	Santander	España
4	Pérez	Juan	Málaga	España

3.5 Borrados y modificaciones, integridad referencial. Cambios en cascada

El mayor inconveniente cuando realizamos cambios en los datos mediante el **borrado y la modificación** de registros es la integridad referencial. Debemos tener en cuenta que las bases de datos siguen un modelo relacional, lo que significa que las tablas están interconectadas entre sí.

Si eliminásemos un registro al que hace referencia otra tabla, no haríamos más que dejar dicha tabla "**huérfana**", lo que pondría en peligro la integridad estructural de la base de datos. Para solucionar este problema, deberíamos repasar todas las tablas y modificar esos registros, lo que supondría un insigne trabajo e, incluso, en bases de datos de gran tamaño, resultaría inviable.



Estructura del modelo relacional y la dependencia de datos.

Por esta razón, los **sistemas gestores** ofrecen la función que permite realizar los cambios en cascada, una tarea que consiste en analizar todas las tablas de la base de datos y aplicarles, de manera automática, los cambios necesarios a aquellos registros que hagan referencia a los datos modificados.

Para que esto sea posible, es necesario establecer correctamente las relaciones entre claves primarias y claves foráneas entre las tablas.

En el siguiente ejemplo vemos como la tabla padre tiene la clave primaria id "PRIMARY KEY (id)" y que la tabla hijo tiene la instrucción "FOREIGN KEY (padre_id)" que indica que su atributo padre_id está vinculado a la id de la tabla padre "REFERENCES padre(id)".

Esto significa que, si eliminamos el registro padre (o modifi camos la ID), existirá un registro hijo que quedará huérfano y se producirá un error de integridad referencial. Para solucionarlo, disponemos de las instrucciones ON UPDATE y ON DELETE, que se encargan de modificar automáticamente el registro de la tabla hijo, al realizar modificaciones en la tabla padre. Los cambios pueden ser:

```
FOREIGN KEY [id] (nombre_índice, ...)
REFERENCES nombre_de_tabla (nombre_índice, ...)
[ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
[ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
```

Dichos cambios soportan los siguientes modifi cadores:

- **CASCADE**: modifica el valor de la clave foránea por el nuevo valor.
- **SET NULL**: modifica la clave foránea y pone el valor NULL.
- **NO ACTION**: omite la sentencia ON UPDATE y ON DELETE, con lo que no se realiza ninguna acción.
- **RESTRICT**: es equivalente a NO ACTION.

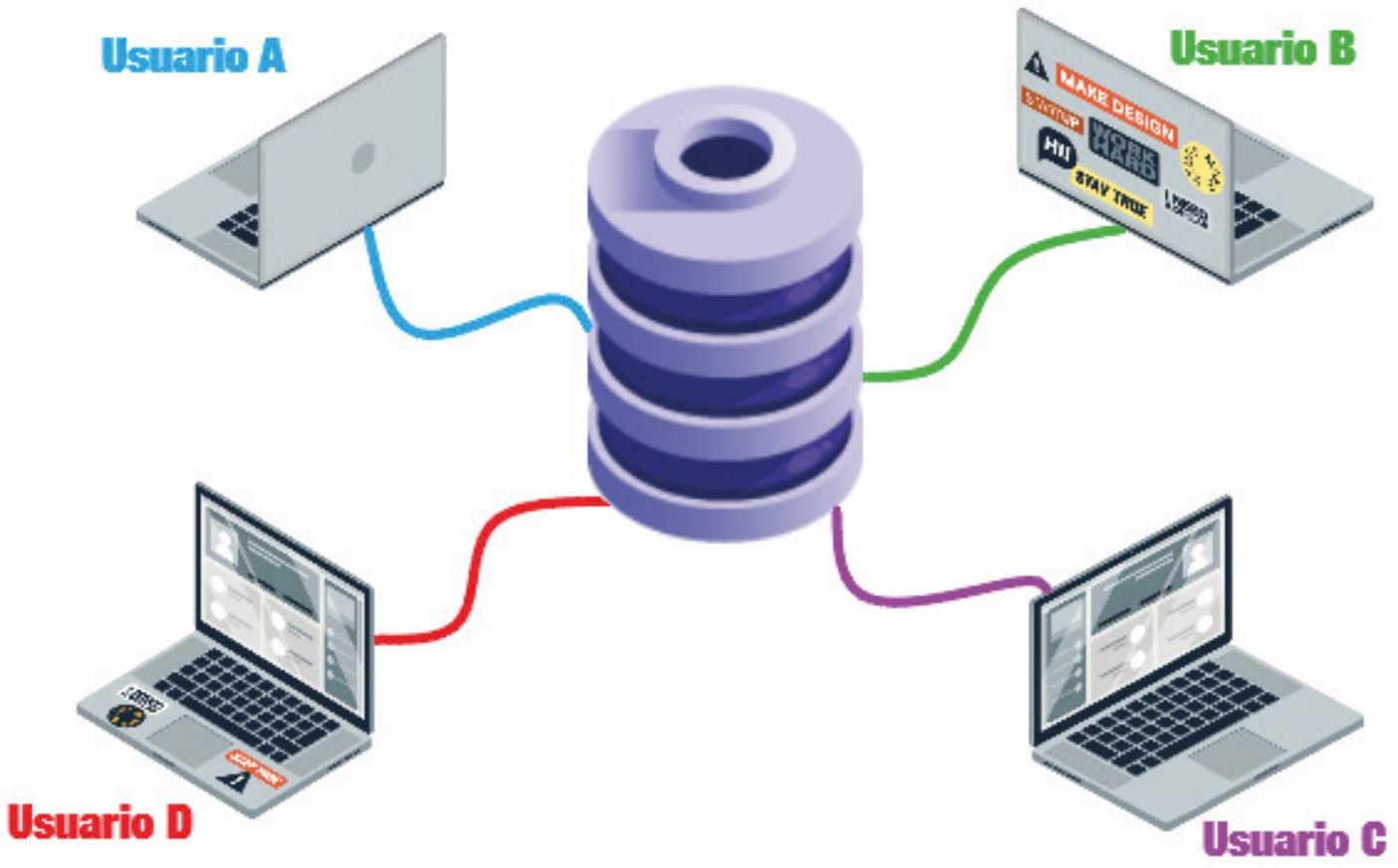
```
CREATE TABLE padre(
    id INT NOT NULL,
    PRIMARY KEY (id)
    ) ENGINE=INNODB;

CREATE TABLE child(
    id INT,
    padre_id INT,
    nombre CHAR
    FOREIGN KEY (padre_id)
      REFERENCES padre(id)
      ON UPDATE CASCADE ON DELETE CASCADE
    ) ENGINE=INNODB;
```

3.6 Subconsultas y composiciones en órdenes de edición

Las **órdenes de edición** son ejecuciones de consultas realizadas en el núcleo del servidor sin necesidad de implicar al usuario. Se ejecutan al cumplir cierta condición definida previamente y suelen utilizarse para el mantenimiento de los datos; por ejemplo, eliminar una tabla de datos temporales al llegar a un número determinado de registros.

3.7 Transacciones. Sentencias de procesamiento de transacciones

Las **transacciones** son unidades de trabajo independiente. Si una unidad de trabajo se realiza con éxito, todas las modificaciones de datos realizadas durante la transacción se harán efectivas. Esto es especialmente útil para el control de errores: en un proceso en el que se realizan varias operaciones, se utiliza el sistema de transacciones con el fin de que, en el caso de que se produzca algún error durante el proceso, todos los datos vuelvan a su estado inicial.



Acceso simultáneo a los datos.

Un ejemplo claro es una transacción bancaria, en la que antes de descontar el dinero de una cuenta origen hay que confirmar que se ha ingresado en el banco de destino:

```
BEGIN TRANSACTION transfer2;
INSERT INTO Cuenta2 VALUES(100);
INSERT INTO Cuenta1 VALUES(-100);
ROLLBACK;
COMMIT TRANSACTION transfer2;
BEGIN TRANSACTION transfer1;
INSERT INTO Cuenta2 VALUES(100);
INSERT INTO Cuenta1_noexiste VALUES(-100);
*Comprobación de Cuenta1_noexiste
**Si hay algun error se deshace únicamente transfer1
ROLLBACK;
```

Al ejecutar esta sentencia se realizará la transfer2 que ha sido confirmada mediante la sentencia COMMIT antes de ejecutar el ROLLBACK , mientras que la transfer1 no se ejecutará debido a que la cuenta destino no existe.

3.8 Problemas asociados al acceso simultáneo a los datos

Un problema común en el tratamiento de datos es el acceso simultáneo. Los usuarios que están modificando datos pueden afectar a otros usuarios que están leyendo o modificando dichos datos. Veamos cómo:

- **Mediante actualizaciones perdidas.** Sucede cuando varios usuarios modifican los mismos datos. El sistema sólo guardará la última modificación realizada y perderá el resto.
- **Mediante una dependencia no confirmada.** Sucede cuando un usuario lee un dato que está siendo modificado por una transacción. Es posible que la transacción sea errónea y devuelva el valor a su estado original.

Un ejemplo claro al respecto podemos verlo en las páginas web de reserva de hoteles, en las que varias personas pueden ver que la Suite Nupcial está libre e iniciar la reserva, pero al terminar de rellenar los datos el sistema impide que se finalice el proceso porque otro usuario ha realizado la reserva con anterioridad.

Si no se controlase esta situación, la última persona que ha reservado la habitación sustituiría la reserva de la primera, produciéndose un caso de actualización perdida.

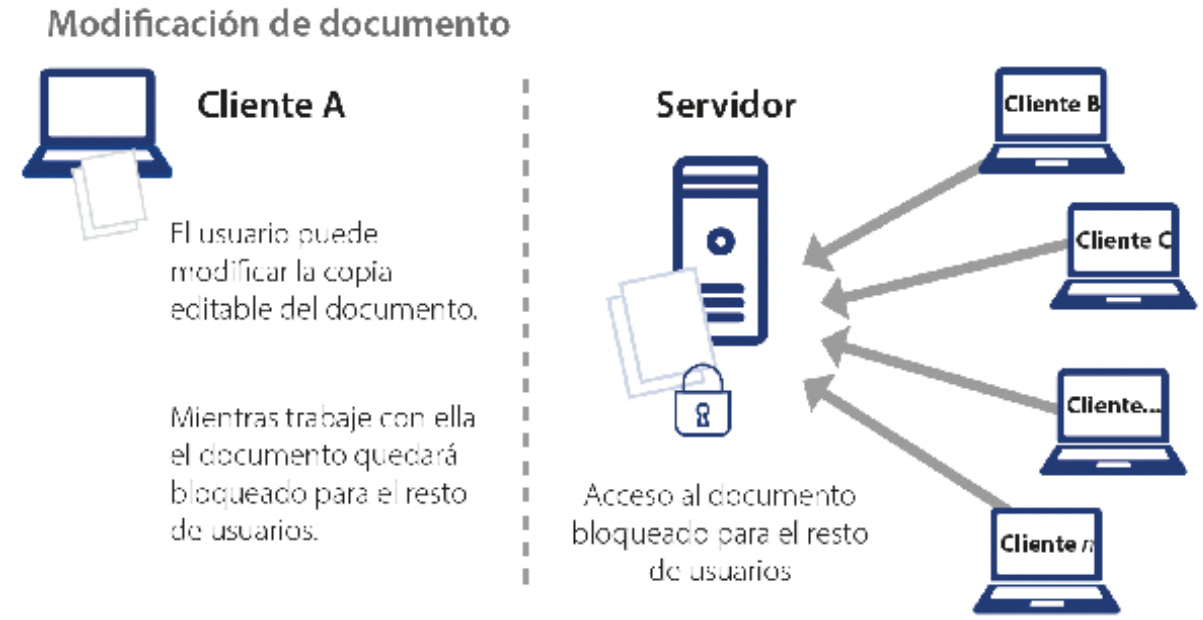
3.9 Bloqueos compartidos y exclusivos. Políticas de bloqueo

En las bases de datos es habitual que varios usuarios accedan a los mismos datos simultáneamente. Por ello, hay que tener en cuenta qué políticas de acceso debemos aplicar en el caso de realizar modificaciones sin que afecten al resto de usuarios, es decir, tenemos que decidir cuál es nuestra política de gestión de bloqueos. Veamos cuáles son:

- **Bloqueo compartido.** Se utiliza para operaciones de lectura que no cambian ni actualizan datos, como la instrucción SELECT.
- **Bloqueo exclusivo**. Se utiliza para operaciones de modificación de datos, como INSERT, UPDATE o DELETE. Garantiza que no puedan realizarse varias actualizaciones simultáneamente en el mismo recurso.

Podemos dividir las políticas de bloqueo de acceso en dos tipos:

- **Control de simultaneidad optimista.** Las operaciones de lectura no bloquean las operaciones de modificación de datos; es decir, un usuario puede modificar los datos aunque otro los esté leyendo en ese momento.
- **Control de simultaneidad pesimista.** Es aquel que garantiza que las operaciones de lectura tengan acceso a los datos actuales y los bloques para que no puedan ser modificados por otros usuarios



Restricción a los otros usuarios por parte del cliente A mientras modifica los datos.

Si tomamos el ejemplo anterior, un control de simultaneidad pesimista sería el ideal para asegurarse de que otros usuarios no puedan reservar la Suite Nupcial una vez iniciado el propio proceso de reserva. Al realizar una lectura de la información de la Suite y hacer clic en "Reservar", el sistema creará un bloqueo en ese registro que no permitirá que nadie más modifique los datos. Una vez hecha la reserva o cerrada la página, el bloqueo terminará.

4. Resumen

Como en el caso de la obtención de datos, existen consultas para realizar modificaciones en las bases de datos, como la inserción o eliminación de registros. La inserción de un nuevo registro se lleva a cabo mediante la sentencia INSERT, en la que estableceremos la tabla en la que deseamos introducir los datos, así como cada una de las columnas con su respectivo valor.

La modificación se realiza con la consulta UPDATE, y al igual que con la sentencia INSERT, será necesario definir la tabla y cada una de las columnas a las que se desea aplicar la modificación, así como también el nuevo valor.

Hay que tener muy en cuenta que, además de la tabla a la que pertenece el registro, deberemos introducir una condición para seleccionar el registro que será modificado, ya que de lo contrario se realizarían sobre todos los registros. En estos casos, es especialmente útil una clave primaria que identifique inequívocamente cada fila de la tabla.

La eliminación se lleva a cabo mediante la consulta DELETE, en la que, como en la modificación, deberemos especificar mediante una cláusula condicional el registro que deseamos borrar.

Generalmente, a las bases de datos acceden varios usuarios, por lo que hay que tener en cuenta las políticas de restricción y uso exclusivo a la hora de realizar modificaciones en los datos. De esta manera evitaremos errores de acceso en el momento de realizar la modificación.

Por último, deberemos tener en cuenta que la naturaleza de las bases de datos relacionales implica que varias tablas, y por lo tanto sus datos, estén interrelacionadas, por lo que es posible borrar un registro que esté siendo referenciado desde otra tabla; si esto ocurre se genera un error de integridad referencial.

Para solucionar estas situaciones, las bases de datos ofrecen la opción de los cambios en cascada con los que, al realizarse un cambio en un registro referenciado, la modificación es aplicada a todas las demás tablas de manera automática.